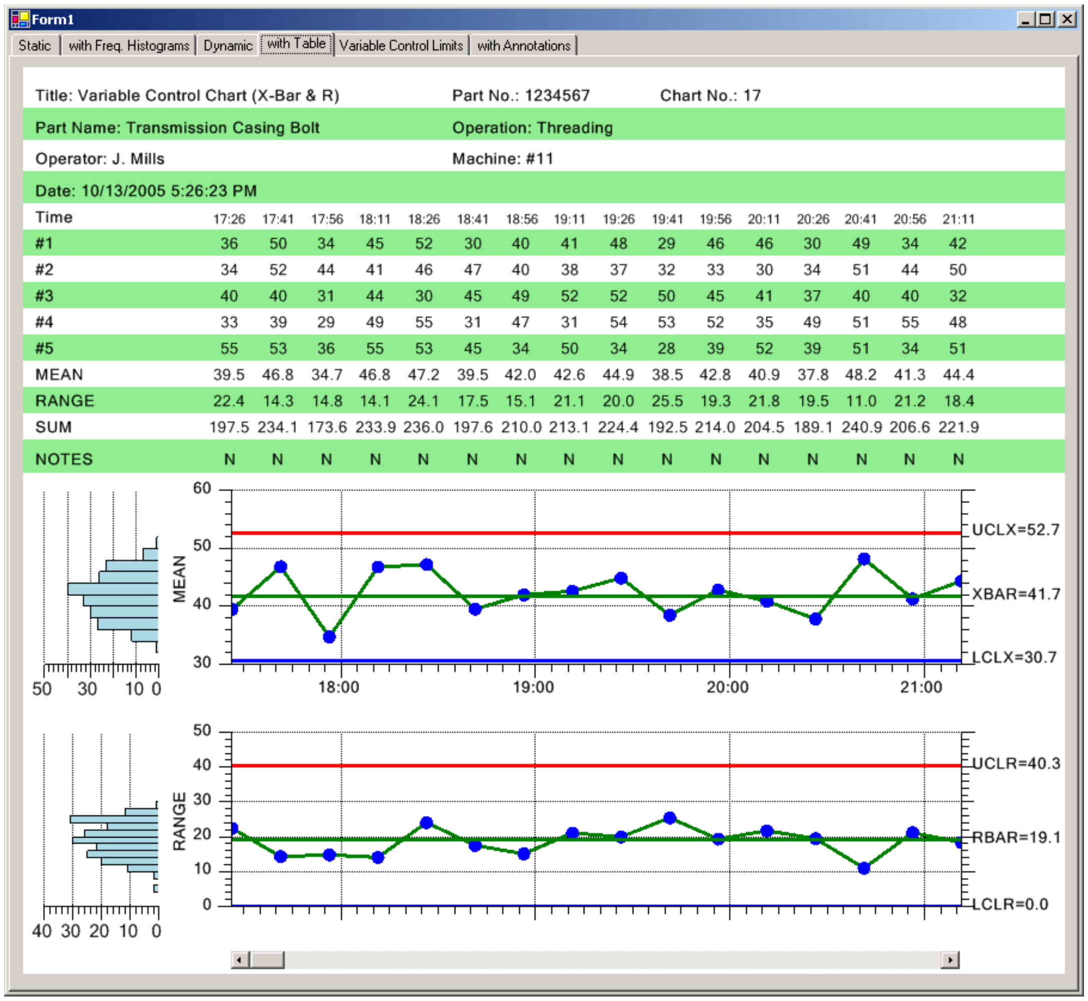
QCSPCChart SPC Control Chart Tools for Java



Title: Variable Control Chart (X-Bar & R) Part No.: 1234567 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Mills Machine: #11

Date: 10/13/2005 5:26:23 PM

Time	17:26	17:41	17:56	18:11	18:26	18:41	18:56	19:11	19:26	19:41	19:56	20:11	20:26	20:41	20:56	21:11
#1	36	50	34	45	52	30	40	41	48	29	46	46	30	49	34	42
#2	34	52	44	41	46	47	40	38	37	32	33	30	34	51	44	50
#3	40	40	31	44	30	45	49	52	52	50	45	41	37	40	40	32
#4	33	39	29	49	55	31	47	31	54	53	52	35	49	51	55	48
#5	55	53	36	55	53	45	34	50	34	28	39	52	39	51	34	51
MEAN	39.5	46.8	34.7	46.8	47.2	39.5	42.0	42.6	44.9	38.5	42.8	40.9	37.8	48.2	41.3	44.4
RANGE	22.4	14.3	14.8	14.1	24.1	17.5	15.1	21.1	20.0	25.5	19.3	21.8	19.5	11.0	21.2	18.4
SUM	197.5	234.1	173.6	233.9	236.0	197.6	210.0	213.1	224.4	192.5	214.0	204.5	189.1	240.9	206.6	221.9
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Contact Information

Company Web Site: http://www.quinn-curtis.com

Electronic mail

General Information: info@quinn-curtis.com
Sales: sales@quinn-curtis.com
Technical Support Forum

http://www.quinn-curtis.com/ForumFrame.htm

Revision Date 1/6/2014 Rev. 2.3

Quinn-Curtis, Inc. Tools Tools for *Java* END-USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Software End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Quinn-Curtis, Inc. for the Quinn-Curtis, Inc. SOFTWARE identified above, which includes all Quinn-Curtis, Inc ***Java*** software (on any media) and related documentation (on any media). By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE. If the SOFTWARE was mailed to you, return the media envelope, UNOPENED, along with the rest of the package to the location where you obtained it within 30 days from purchase.

1. The SOFTWARE is licensed, not sold.

2. GRANT OF LICENSE.

(A) Developer License. After you have purchased the license for SOFTWARE, and have received the file containing the licensed copy, you are licensed to copy the SOFTWARE only into the memory of the number of computers corresponding to the number of licenses purchased. The primary user of the computer on which each licensed copy of the SOFTWARE is installed may make a second copy for his or her exclusive use on a portable computer. Under no other circumstances may the SOFTWARE be operated at the same time on more than the number of computers for which you have paid a separate license fee. You may not duplicate the SOFTWARE in whole or in part, except that you may make one copy of the SOFTWARE for backup or archival purposes. You may terminate this license at any time by destroying the original and all copies of the SOFTWARE in whatever form.

(B) 30-Day Trial License. You may download and use the SOFTWARE without charge on an evaluation basis for thirty (30) days from the day that you DOWNLOAD the trial version of the SOFTWARE. The termination date of the trial SOFTWARE is embedded in the downloaded SOFTWARE and cannot be changed. You must pay the license fee for a Developer License of the SOFTWARE to continue to use the SOFTWARE after the thirty (30) days. If you continue to use the SOFTWARE after the thirty (30) days without paying the license fee you will be using the SOFTWARE on an unlicensed basis.

Redistribution of 30-Day Trial Copy. Bear in mind that the 30-Day Trial version of the SOFTWARE becomes invalid 30-days after downloaded from our web site, or one of our sponsor's web sites. If you wish to redistribute the 30-day trial version of the SOFTWARE you should arrange to have it redistributed directly from our web site If you are using SOFTWARE on an evaluation basis you may make copies of the evaluation SOFTWARE as you wish; give exact copies of the original evaluation SOFTWARE to anyone; and distribute the evaluation SOFTWARE in its unmodified form via electronic means (Internet, BBS's, Shareware distribution libraries, CD-ROMs, etc.). You may not charge any fee for the copy or use of the evaluation SOFTWARE itself. You must not represent in any way that you are selling the SOFTWARE itself. You must distribute a copy of this EULA with any copy of the SOFTWARE and anyone to whom you distribute the SOFTWARE is subject to this EULA.

(C) Redistributable License. The standard Developer License permits the programmer to deploy and/or distribute applications that use the Quinn-Curtis SOFTWARE, royalty free. We cannot allow developers to use this SOFTWARE to create a graphics toolkit (a library or any type of graphics component that will be used in combination with a program development environment) for resale to other developers.

If you utilize the SOFTWARE in an application program, or in a web site deployment, should we ask, you must supply Quinn-Curtis, Inc. with the name of the application program and/or the URL where the SOFTWARE is installed and being used.

3. RESTRICTIONS. You may not reverse engineer, de-compile, or disassemble the SOFTWARE, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this

Should you have any questions concerning this EULA, or if you desire to contact Quinn-Curtis, Inc. for any reason, please contact Quinn-Curtis, Inc. by mail at: Quinn-Curtis, Inc., 18 Hearthstone Dr., Medfield MA 02052 USA, or by telephone at: (508)359-6639, or by electronic mail at: support@Quinn-Curtis.com

Table of Contents

1. Introduction....................1
   What's New in Rev. 2.3....................1
   What's New in Rev. 2.2....................2
   New Features found in the 2.1 version of QCSPCChart....................2
   New Features found in the 2.0 version of QCSPCChart....................2
   New Features added to the 1.7 version of QCSPCChart....................4
   Tutorials....................5
   Customer Support....................5
   SPC Control Chart Tools for Java Background....................5
   Quinn-Curtis SPC (Statistical Process Control) Software....................7
   QCSPCChart for Java Dependencies....................9
   SPC Control Chart Tools for Java Dependencies....................10
   Directory Structure of QCSPCChart for Java....................10
   Chapter Summary....................13
2. Standard SPC Control Charts 15
   Variable Control Charts....................16
   Attribute Control Charts....................31
   Other Important SPC Charts....................36
3. Class Architecture of the SPC Control Chart Tools for Java Class Library 41
   Major Design Considerations....................41
   SPC Control Chart Tools for Java Class Summary....................43
   SPC Control Chart Tools for Java Class Hierarchy....................44
   QCSPCChart Classes....................45
4. QCChart2D for Java Class Summary....................58
   Chart Window Classes....................59
   Data Classes....................59
   Scale Classes....................60
   Coordinate Transform Classes....................61
   Auto-Scaling Classes....................63
   Chart Object Classes....................64
   Mouse Interaction Classes....................88
   File and Printer Rendering Classes....................89
   Miscellaneous Utility Classes....................90
5. SPC Control Data and Alarm Classes....................94
   Class SPCControlChartData 94
   Control Limit Alarms....................109
   Control Limit Alarm Event Handling....................112
   SPCSampledValueRecord 115
   SPCCalculatedValueRecord 115
   SPCProcessCapabilityRecord 116
   SPCGeneralizedTableDisplay....................116
6. SPC Variable Control Charts....................117

Time-Based and Batch-Based SPC Charts....................119
Creating a Batch-Based Variable Control Chart....................179
Changing the Batch Control Chart X-Axis Labeling Mode....................182
Changing Default Characteristics of the Chart....................186
7. SPC Attribute Control Charts....................190
Time-Based and Batch-Based SPC Charts....................191
Changing the Batch Control Chart X-Axis Labeling Mode....................235
8. Named and Custom Control Rule Sets....................241
Western Electric (WECO) Rules....................241
Western Electric (WECO) Rules....................242
Nelson Rules....................243
AIAG Rules....................243
Juran Rules....................243
Hughes Rules....................244
Gitlow Rules....................244
Duncan Rules 244
Westgard Rules 245
Control Rule Templates....................245
Implementing a Named Rule Set....................250
Modifying Existing Named Rules....................254
Creating Custom Rules Sets Based on Named Rules....................255
Creating Custom Rules Sets Based on a Template....................257
Creating Custom Rules Not Associated With Sigma Levels....................258
Control Limit Alarm Event Handling....................265
9. Frequency Histogram, Pareto Diagram and Normal-Probability Charts....................267
Frequency Histogram Chart....................267
Probability Plots....................275
Pareto Diagrams....................279
10. File and Printer Rendering Classes....................284
Printing a Chart....................285
Capturing the Chart as a Buffered Image....................286
11. Regionalization for non-USA English Markets....................289
12. Compiling and Running the Example Programs....................303
Using Eclipse, NetBeans and JBuilder IDE's....................303
Eclipse....................303
NetBeans....................311
Borland JBuilder 2005 Developers Edition....................320
13. Tutorial – Creating QCSPCChart for Java Applications and Applets....................327
Java Applications....................327
Standalone (without a compiler) Java Applications....................334
Java Applets....................335

SPC Control Chart Tools for *Java*

1. Introduction

What's New in Rev. 2.3

All of the new features found in the 2.3 version of QCSPCChart were added to QCChart2D - primarily a new collection of event based charting classes. None of these routines have been used in the QCSPCChart software.

Event-Based Charting
A new set of classes have been added in support of new, event-based plotting system. In event-based plotting, the coordinate system is scaled to the number of event objects. Each event object represents an x-value, and one or more y-values. The x-value can be time based, or numeric based, while the y-values are numeric based. Since an event object can represent one or more y-values for a single x-value, it can be used as the source for simple plot types (simple line plot, simple bar plot, simple scatter plot, simple line marker plot) and group plot types (open-high-low-close plots, candlestick plots, group bars, stacked bars, etc.). Event objects can also store custom data tooltips, and x-axis strings. The most common use for event-based plotting will be for displaying time-based data which is discontinuous: financial markets data for example. In financial markets, the number trading hours in a day may change, and the actual trading days. Weekends, holidays, and unused portions of the day can be excluded from the plot scale, producing continuous plots of discontinuous data. The following classes have been added to the software in support of event-based charting.

- **ChartEvent** - A ChartEvent object stores the position value, the time stamp, y-values, and custom strings associated with the event.
- **EventArray** - A utility array class used to store ChartEvent objects
- **EventAutoScale** – An auto-scale class used by the EventCoordinates class.
- **EventAxis** - Displays an axis based on an EventCoordinates scale
- **EventAxisLabels** – Displays the string labels labeling the tick marks of an EventAxis
- **EventCoordinates** – Event coordinates define a coordinate system based on the the attached Event datasets
- **EventGroupDataset** – A group dataset which uses ChartEvent objects as the source of the data. It is used to feed data into the group plotting routines.
- **EventScale** – An event scale class used to convert between event coordinates and device coordinates.
- **EventSimpleDataset** - A simple dataset which uses ChartEvent objects as the source of the data. It is used to feed data into the simple plotting routines.

What's New in Rev. 2.2

- Three new control charts – Moving Average/Moving Range (MAMR), Moving Average/Moving Sigma (MAMS) and the Number Defects per Million (DPMO). See Chapter 6 (SPC Variable Control Charts) and Chapter 7 (*SPC Attribute Control Charts*).
- Direct support for additional control rule sets – Nelson, AIAG, Juran, Hughes, Gitlow, Westgard, and Duncan, in addition to the WE (Western Electric) and Supplemental Rules. See Chapter 8 (*Named and Custom Control Rule Sets*).
- Create custom sets of control rules, using any rule from any of the standard named control rule sets. See Chapter 8 (*Named and Custom Control Rule Sets*).
- Define custom control rules: based any of our predefined control rule templates See Chapter 8 (*Named and Custom Control Rule Sets*).
- Regionalization – All strings used in the software have been moved to a static class which can be initialized at runtime with country specific strings. See Chapter 11 (*Regionalization for non-USA English Markets*).

New Features found in the 2.1 version of QCSPCChart

Revision 2.1 adds the following new features:

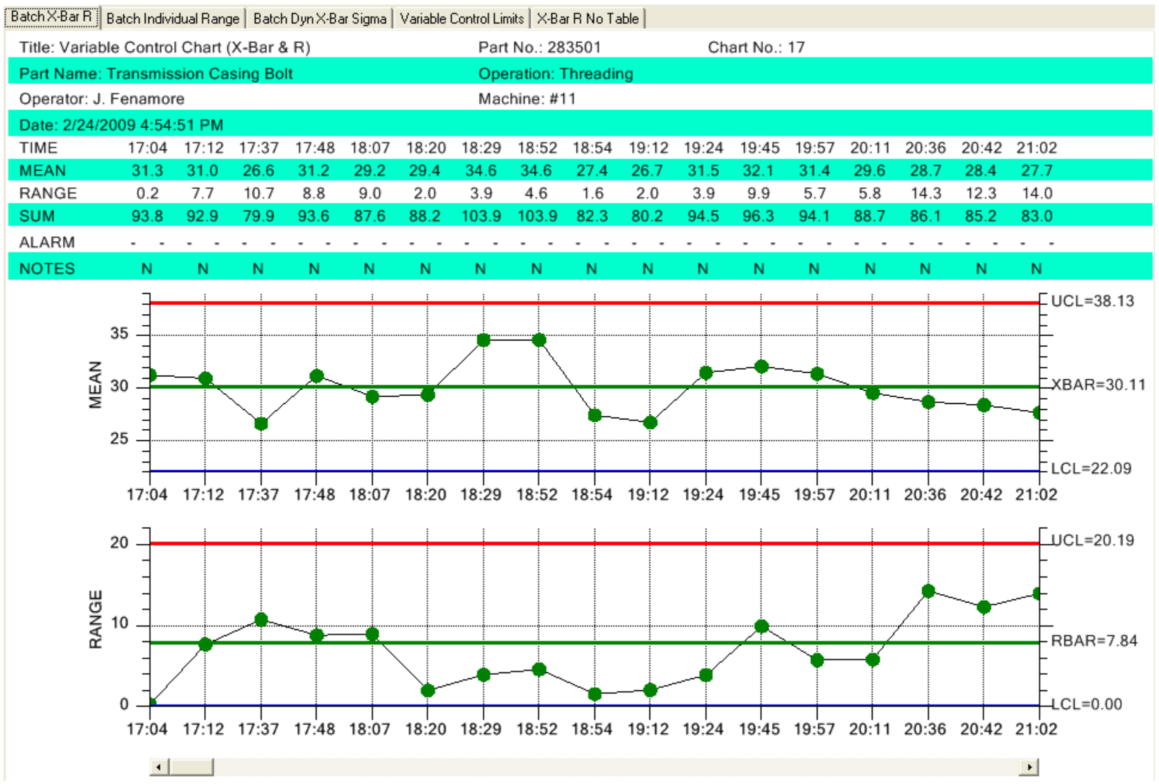
- Specification Limits – Unique from control limits. See page 162.
- Western Electric Trending (or Supplemental) Rules (Rules 5 – 8). See page 158.
- Simpler way to set control limits for +-1, 2 and 3 sigma - Add3SigmaControlLimits – See page 155.
- Solid area fill between limit lines - ControlLimitLineFillMode – See page 157.

New Features found in the 2.0 version of QCSPCChart

Revision 2.0 follows Revision 1.7. Most new features associated with revision 2.0 are part of the QCChart2D software, on top of which the QCSPCChart software is built. As far this software goes, only a few featues specific to Revision 2.0 of QCSPCChart have been added. These include:

⑤ The batch control chart templates (SPCBatchVariableControlChart, SPCBatchAttributeControlChart) have new x-axis labeling modes. Label the x-axis tick marks using a batch number (the original and default mode), a time stamp, or a user-defined string.

⑤ The x-axis labels can be rotated 360 degrees.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Fenamore Machine: #11

Date: 2/24/2009 4:54:51 PM

TIME	17:04	17:12	17:37	17:48	18:07	18:20	18:29	18:52	18:54	19:12	19:24	19:45	19:57	20:11	20:36	20:42	21:02
MEAN	31.3	31.0	26.6	31.2	29.2	29.4	34.6	34.6	27.4	26.7	31.5	32.1	31.4	29.6	28.7	28.4	27.7
RANGE	0.2	7.7	10.7	8.8	9.0	2.0	3.9	4.6	1.6	2.0	3.9	9.9	5.7	5.8	14.3	12.3	14.0
SUM	93.8	92.9	79.9	93.6	87.6	88.2	103.9	103.9	82.3	80.2	94.5	96.3	94.1	88.7	86.1	85.2	83.0
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The time stamp of batch control chart does not have to be does not have to have an equal time spacing between adjacent sample groups.

Revision 2.0 has added many new to QCChart2D. New features include:

- Five new plot types: **BoxWiskerPlot**, **FloatingStackedBarPlot**, **RingChart**, **SimpleVersaPlot** and **GroupVersaPlot**
- Elapsed time scaling to compliment the time/date scaling. Includes a set of classes specifically for elapsed time charts: **ElapsedTimeLabel**, **ElapsedTimeAutoScale**, **ElapsedTimeAxis**, **ElapsedTimeAxisLabels**, **ElapsedTimeCoordinates**, **ElapsedTimeScale**, **ElapsedTimeSimpleDataset** and **ElapsedTimeGroupDataset**.
- Vertical axis scaling for time/date and elapsed time
- A **DatasetViewer** class for the grid-like display of dataset information in a table.
- A **MagniView** class: a new way to zoom data
- A **CoordinateMove** class – used to pan the coordinate system, left, right, up, down.
- Zoom stack processing is now internal to the ChartZoom class

Refer to the QCChart2D manual for information specific to these new features.

New Features added to the 1.7 version of QCSPCChart.

A large number of new features were added to QCSPCChart in the change from Revision 1.6 to 1.7. These are summarized below:

Eliminated the QCLicense License Fle

We have eliminated the QCLicense license file from the software and with it the need to purchase additional Redistributable Licenses. Once you purchase the software, you the developer, can create application programs that use this software and redistribute the programs and our libraries royalty free. As a development tool, i.e. using this software in conjunction with a compiler, the software is still governed by a single user license and cannot be shared by multiple individuals unless additional copies, or a site license, have been purchased.

New SPC Chart Types and Features

⑤ Three new charts have been added to the software: **EWMA** (Exponentially Weighted Moving Average), **MA** (Moving Average), and **CuSum** (Cumulative Sum) charts.

⑤ Three variants of existing charts have been adapted to support variable subgroup sample sizes: X-Bar Sigma, p-Chart (Fraction or Percent of Defective Parts) and u-Chart (Number of Defects per Unit).

⑤ The FrequencyHistogramChart now includes optional limit lines, and an optional fit and overlay of a normal curve to the frequency data.

New Alarm Features

⑤ There is a new status line in the table section that gives a direct indication of whether or not the corresponding process variable is in, or out of, alarm. The status line has a tooltip, so if you click on an alarm, a pop-up box will show you details.

⑤ Optionally, the entire column associated with a sample interval can be color highlighted to indicate an alarm condition.

⑤ Alarm details can be automatically logged to the notes log.

⑤ The symbol used to plot a process variable point in the primary and secondary charts can be made to change color in the event of an alarm.

Tutorials

Chapter 13 is a tutorial that describes how to get started running the example programs that come with the **SPC Control Chart Tools for *Java*** charting software. Chapter 11 is a tutorial that describes how to use the software to create charts for Java applications and applets.

Customer Support

Use our forums at http://www.quinn-curtis.com/ForumFrame.htm for customer support. Please, do not post questions on the forum unless you are familiar with this manual and have run the examples programs provided. We try to answer most questions by referring to the manual, or to existing example programs. We will always attempt to answer any question that you may post, but be prepared that we may ask you to create, and send to us, a simple example program. The program should reproduce the problem with no, or minimal interaction, from the user. You should strip out of any code not directly associated with reproducing the problem. You can use either your own example or a modified version of one of our own examples.

SPC Control Chart Tools for *Java* Background

In a competitive world environment, where there are many vendors selling products and services that *appear* to be the same, **quality**, both real and perceived, is often the critical factor determining which product wins in the marketplace. Products that have a reputation for higher quality command a premium, resulting in greater market share and profit margins for the manufacturer. Low quality products not only take a big margin hit at the time of sale, but also taint the manufacturer with a reputation that will hurt future sales, regardless of the quality of future products. Users have a short memory. A company's quality reputation is only as good as the quality of its most recent product.

The measurement, control and gradual improvement of quality is the goal of all quality systems, no matter what the name. Some of the more common systems are known as **SCC** (Statistical Quality Control) **Quality Engineering**, **Six-Sigma**, **TQM** (Total Quality Management), **TQC** (Total Quality Control), **TQA** (Total Quality Assurance) and **CWQC** (Company- Wide Quality Control). These systems work on the principle that management must integrate quality into the basic structure of the company, and not relegate it to a Quality Control group within the company. Historically, most of the innovations in quality systems took place in the 20th century, with pioneering work carried out by Frederick W. Taylor (Principles of Scientific Management), Henry Ford (Ford Motor), W. A. Shewhart (Bell Labs), W. E. Deming (Department of Agriculture, War department, Census Bureau), Dr. Joseph M. Juran (Bell Labs), and Dr. Armand V.

Feigenbaum among others. Most quality control engineers are familiar with the story of how the statistical quality control pioneer, W. E. Deming, frustrated that US manufacturers only gave lip service to quality, took the quality system he developed to Japan, where it was embraced with almost religious zeal. Japananese industry considers Deming a national hero, where his quality system played a major role in the postwar expansion of the Japanese economy. Twenty to thirty years after Japan embraced his methods, Deming found a new audience for his ideas at US companies that wanted to learn *Japanese* methods of quality control.

All quality systems use Statistical Process Control (**SPC**) to one degree or another. SPC is a family of statistical techniques used to track and adjust the manufacturing process in order to produce gradual improvements in quality. While it is based on sophisticated mathematical analysis involving sampling theory, probability distributions, and statistical inferences, SPC results can usually be summarized using simple charts that even management can understand. SPC charts can show how product quality varies with respect to critical factors that include things like batch number, time of day, work shift personal, production machine, and input materials. These charts have odd names like X-Bar R, Median Range, Individual Range, Fraction Number Non-Conforming, and NP. The charts plot some critical process variable that is a measurement of product quality and compares it to predetermined limits that signify whether or not the process is working properly.

Initially, quality control engineers create all SPC charts by hand. Data points were painstakingly gathered, massaged, summed, averaged and plotted by hand on graph paper. It is still done this way in many cases. Often times it is done by the same factory floor personal who control the process being measured, allowing them to "close the loop" as quickly as possible, correcting potential problems in the process before it goes out of control. Just as important, SPC charts tell the operator when to leave the process alone. Trying to micro-adjust a process, when the process is just exhibiting normal random fluctuations in quality, will often drive the process out of control faster than leaving it alone.

The modern tendency is to automate as much of the SPC chart creation process as possible. Electronic measuring devices can often measure quality in real-time, as items are coming off the line. Usually some form of sampling will be used, where one of every N items is measured. The sampled values form the raw the data used in the SPC chart making process. The values can be entered by hand into a SPC chart making program, or they can be entered directly from a file or database connection, removing the potential for transcription errors. The program displays the sampled data in a SPC chart and/table where the operator or quality engineer can make a judgment about whether or not the process is operating in or out of control.

Usually the SPC engineer tasked with automating an existing SPC charting application has to make a decision about the amount of programming he wants to do. Does he purchase an application package that implements standard SPC charts and then go about defining the charts using some sort of menu driven interface or wizard. This is probably the most expensive in terms of up front costs, and the least flexible, but the cheapest in development costs since a programmer does not have to get involved creating the displays. Another choice is to use a general purpose spreadsheet package with charting capability to record, calculate, and display the charts. This is probably a good choice if your charting needs are simple, and you are prepared to write complicated formulas as spreadsheet entries, and your data input is not automated. Another choice is writing the software from scratch, using a charting toolkit like our ***QCChart2D*** software as the base, and creating custom SPC charts using the primitives in the toolkit. This is cheaper up front, but may be expensive in terms of development costs. Often times the third option is the only one available because the end-user has some unique requirement that the pre-packaged software can't handle, hence everything needs to programmed from scratch.

Quinn-Curtis SPC (Statistical Process Control) Software

We have created a library of SPC routines that represents an intermediate solution. Our SPC software still requires an intermediate level programmer, but it does not require advanced knowledge of SPC or of charting. Built on top our ***QCChart2D,*** it implements templates and support classes for the following SPC charts and control limit calculations.

Variable Control Charts Templates

- Fixed sample size subgroup control charts
    - X-Bar R – (Mean and Range Chart)
    - X-Bar Sigma (Mean and Sigma Chart)
    - Median and Range (Median and Range Chart)
    - X-R (Individual Range Chart)
    - EWMA (Exponentially Weighted Moving Average Chart)
    - MA (Moving Average Chart)
    - MAMR (Moving Average / Moving Range Chart)
    - MAMS (Moving Average / Moving Sigma Chart)
    - CuSum (Tabular Cumulative Sum Chart)
- Variable sample size subgroup control charts
    - X-Bar Sigma (Mean and Sigma Chart)

Attribute Control Charts Templates

- Fixed sample size subgroup control charts
    - p Chart (Fraction or Percent of Defective Parts)
    - np Chart (Number of Defective Parts)
    - c-Chart (Number of Defects)
    - u-Chart (Number of Defects per Unit)
    - Number Defects per Million (DPMO)
- Variable sample size subgroup control charts
    - p Chart (Fraction or Percent of Defective Parts)
    - u-Chart (Number of Defects per Unit)

Analysis Chart Templates

- Frequency Histograms

Probability Charts
Pareto Charts

SPC Support Calculations

Array statistics (sum, mean, median, range, standard deviation, variance, sorting)

SPC Control Limit Calculations

High and low limit control calculations for X-Bar R, X-Bar Sigma, Median and Range, X-R, p, np, c and u charts

SPC Process Capability Calculations

Variable Control Charts include Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk process capability statistics

SPC Control Named Rule Sets

Western Electric (WECO) Runtime and Supplemental Rules
Nelson
AIAG
Juran
Hughes
Gitlow
Duncan
Westgard

The **SPC Control Chart Tools for Java** is a family of templates that integrate the **QCChart2D** charting software with tables, data structures and specialized rendering routines used for the static and dynamic display of SPC charts. The SPC chart templates are pre-programmed classes that create, manage and display the graphs and tables corresponding to major SPC control chart types. Each template can be further customized using method and properties. The programmers can customize the plot objects created in the template, allowing tremendous flexibility in the look of the SPC charts.

Like the **QCChart2D** software, the **SPC Control Chart Tools for** Java uses the graphics features found in the Java API. Starting with Version 1.2, Java has a set of graphics classes providing programming tools as powerful as the Windows API routines. Features found in the Java 1.2 API and used in **com.quinncurtis.spcchartjava** package include the following.

- Arbitrary line thickness and line styles for all lines.
- Gradients, fill patterns and color transparency for solid objects.
- Generalized geometry support used to create arbitrary shapes
- Printer and image output support
- Improved font support for a large number of fonts, using a variety of font styles, size and rotation attributes.
- Advanced matrix support for handling 2D transformation.

QCSPCChart for Java Dependencies

The **com.quinncurtis.spcchartjava and com.quinncurtis.chart2djava** packages are self-contained. They use only standard classes that ship with the Java 1.2 API. In addition, all components referenced in the software are **lightweight** components, so that they will run on as many platforms as possible. The software uses the major Java packages listed below.

Package java.awt

The **java.awt** package is the core of Java AWT (Abstract Windowing Toolkit). It contains all of the classes that create user interfaces and draw graphics and images. The **java.awt** classes most often used in the Quinn-Curtis **chart2djava** classes are the **graphics**, **graphics2D**, **geom**, **color**, **font**, and **image**. The software also uses two graphics support packages, **java.awt.geom** and **java.awt.print,** extensively.

Class Java.awt.graphics

The **graphics** class is the abstract base class for all graphics. It contains the basic line drawing, text, area fill and color control functions needed to output a graph to an output device.

Class Java.awt.graphics2d

The **graphics2D** class extends the Graphics class to include a more sophisticated set of functions for line drawing, area filling, text placement, color control, and 2D coordinate transformations.

Class java.awt.image

This class provides routines for creating and modifying images.

Class java.awt.color

Provides a class to define colors in terms of their individual RGB (Red, Green, Blue) components.

Class java.awt.font

This class provides routines for defining and manipulating character fonts.

Package java.awt.geom

This package provides the Java 2D classes used to define and perform operations on objects related to two-dimensional geometry.

Package java.awt.print

This package provides classes and interfaces for a general printing API.

Package java.util.*

This package provides classes for tree, list and vector management, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Package java.text.*

This package provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.

Package java.io.*

This package provides classes for file i/o and serialization.

Package javax.swing.event.*

This package provides for the **MouseInputListener** events fired by Swing components.

Package javax.swing

This package provides classes of "lightweight" (all-Java language) components that, in theory if not practice, work the same on all platforms.

SPC Control Chart Tools for *Java* Dependencies

The **SPC Control Chart Tools for *Java*** class library builds on the **QCChart2D** software package. It uses the classes found in that software and standard classes that ship with the Java API. No other software is required.

Directory Structure of QCSPCChart for *Java*

The **SPC Control Chart Tools for** Java class library uses the standard directory structure also used by the **QCChart2D** software. It adds the **spcchartjava** directory structure under the Quinn-Curtis\Java\com\quinn-curtis folder. For a list of the folders specific to **QCChart2D**, see the manual for **QCChart2D**, QCChart2DJavaManual.pdf.

Drive:

Quinn-Curtis\ - Root directory

java\ - Quinn-Curtis Java based products directory

RedistributableFiles\ - Contains the qcchart2djava.jar and qcspcchartjava.jar files that you should use(they do not contain the javadoc files) when redistributing applications that use this library.

docs\ - Quinn-Curtis Java related documentation directory

lib\ - Quinn-Curtis Java related compiled libraries and components directory

com

quinn-curtis

chart2djava\ – QCChart2D specific

spcchartjava\ - QCSPCChart specific

Examples\ - examples directory

FrequencyHistogram – a simple frequency histogram example using the **FrequencyHistogramChart** class

BatchAttributeControlCharts - a collection of batch attribute control charts, including n, np, c, and u charts using the **SPCBatchAttributeControlChart** class.

BatchVariableControlCharts - a collection of batch variable control charts, including X-Bar R, X-Bar Sigma, Median Range, and X-R charts using the **SPCBatchVariableControlChart** class.

TimeAttributeControlCharts - a collection of time attribute control charts, including n, np, c, and u charts using the **SPCTimeAttributeControlChart** class.

TimeVariableControlCharts BatchVariableControlCharts - a collection of time variable control charts, including X-Bar R, X-Bar Sigma, Median Range, and X-R charts using the **SPCTimeVariableControlChart** class.

MiscTimeBasedControlCharts - a collection of time variable control charts, including EWMA, MA and CuSum charts using the **SPCTimeVariableControlChart** class.

MiscBatchBasedControlCharts - a collection of batch variable control charts, including EWMA, MA and CuSum charts using the **SPCTimeVariableControlChart** class.

ProbabilityPlot - a probability chart using the **ProbabilityChart** class.

ParetoDiagram - a Pareto diagram chart using the **ParetoChart** class.

RulesRulesRules - a collection of control charts demonstrating the the use of named (WECO, Nelson, AIAG, Juran, Hughes, Duncan, Westgard, and Gitlow) and custom rule sets.

SPCApplication1 – A simple X-Bar R example program, using **SPCTimeVariableControlChart**, used in the tutorial.

WERulesVariableControlCharts - a collection of using the WE rules with **SPCTimeVariableControlChart** charts, including X-Bar R, X-Bar Sigma, Median Range, and X-R

VariableSampleSizeControlCharts - a collection of the variable control (X-Bar Sigma), and attribute control (p- and u-charts) that support variable sample subgroup sizes.

SPC Control Chart Tools for *Java* class library uses the QCChart2D licensing system. There are two main versions: the 30-day trial version and the developer version. The different versions are summarized below.

30-Day Trial Version

The trial version of **QCSPCChart for *Java*** is downloaded as a zip file named **Trial_QCSPCChartJavaR17x.zip**. The 30-day trial version stops working 30 days after the initial download. The trial version includes a version message in the upper left corner of the graph window that cannot be removed.

Developer Version

The developer version of **QCSPCChart for *Java*** is downloaded as a zip file, name something similar to JAVSPCDEV1R2x0x353x1.zip. The developer version does not time out and you can use it to create application programs that you can distribute royalty free. You can download free updates for a period of 2-years. When you placed your order, you were e-mailed download link(s) that will download the software. Those download links will remain active for at least 2 years and should be used to download current versions of the software. After 2 years you may have to purchase an upgrade to continue to download current versions of the software

When you actually deploy and applet or application, independent of compiler, make sure you use the qcchart2djava.jar and/or the qcspcchartjava.jar libraries found in the Quinn-Curtis\java\RedistributableFiles folder, NOT the same-named ones in the Quinn-Curtis\java\lib folder.

The JAR files in the \lib folder contain the javadoc documentation for the classes, making those the ones you should use for development, though that adds several megabytes their size. The JAR files in the \RedistributableFiles folder have had the javadoc documentation stripped out, leaving only the library classes. This makes the JAR files very, resulting in faster load times, for both applets and applications.

Chapter Summary

The remaining chapters of this book discuss the **SPC Control Chart Tools for *Java*** package designed to run on any hardware that has a Java runtime installed on it.

Chapter 2 presents a summary of the standard SPC control charts that can be created using the software.

Chapter 3 presents the overall class architecture of the **SPC Control Chart Tools for *Java*** and summarizes all of the classes found in the software.

Chapter 4 summarizes the important **QCSPCChart** classes that you must be familiar with in order to customize advanced features of the **SPC Control Chart Tools for *Java*** software.

Chapter 5 describes the classes that hold SPC control chart data and control limit alarms.

Chapter 6 describes how the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes create common variable control charts: X-Bar R, Median and Range, X-Bar Sigma and X-R charts**.**

Chapter 7 describes how the **SPCTimeAttributeControlChart** and **SPCBatchAttributeControlChart** classes create common attribute control charts: p-, np-, c- and u-charts.

Chapter 8 describes how to implement the named control rules (WECO, Nelson, AIAG, Juran, Hughes, Gitlow, Westgard, and Duncan) control rules. It also describes how to implement custom rules sets, and how to define your own rules using our standardized templates

Chapter 9 describes how the **FrequencyHistogramChart**, **ParetoChart** and **ProbabilityChart** classes create ancillary SPC charts.

Chapter 10 describes how to print the SPC charts, and save them to image files.

Chapter 11 describes how to regionalize the software for non-USA English markets.

Chapter 12 is a tutorial that describes how to compile and run the example programs using Eclipse, NetBeans and JBuilder..

Chapter 13 is a tutorial that describes how to use **SPC Control Chart Tools for *Java*** to create Java Applets and Applications.

2. Standard SPC Control Charts

There are many different types SPC control charts. Normally they fall into one of two major classifications: *Variable Control Charts*, and *Attribute Control Charts*. Within each classification, there are many sub variants. Often times the same SPC chart type has two or even three different names, depending on the software package and/or the industry the chart is used in. We have provided templates for the following SPC control charts:

Variable Control Charts

- Fixed sample size subgroup control charts
    - X-Bar R – (Mean and Range Chart)
    - X-Bar Sigma (Mean and Sigma Chart)
    - Median and Range (Median and Range Chart)
    - X-R (Individual Range Chart)
    - EWMA (Exponentially Weighted Moving Average Chart)
    - MA (Moving Average Chart)
    - MAMR (Moving Average / Moving Range Chart)
    - MAMS (Moving Average / Moving Sigma Chart)
    - CuSum (Tabular Cumulative Sum Chart)
- Variable sample size subgroup control charts
    - X-Bar Sigma (Mean and Sigma Chart)

Attribute Control Charts

- Fixed sample size subgroup control charts
    - p Chart (Fraction or Percent of Defective Parts)
    - np Chart (Number of Defective Parts)
    - c-Chart (Number of Defects)
    - u-Chart (Number of Defects per Unit)
    - Number Defects per Million (DPMO)
- Variable sample size subgroup control charts
    - p Chart (Fraction or Percent of Defective Parts)
    - u-Chart (Number of Defects per Unit)

Time-Based and Batch-Based SPC Charts

We have further categorized *Variable Control charts* and *Attribute Control Charts* as either time- or batch- based. While you may not find this distinction in SPC textbooks (we didn't), it makes sense to us as charting experts. Quality engineers use time-based SPC charts when data is collected using a subgroup interval corresponding to a specific time interval. They use batch-based SPC charts when the data subgroup interval is a sequential batch number that does not correspond to a uniform time interval. The major difference in these two types of SPC charts is the display of the x-axis. Variable control charts that sample using a uniform time interval will generally use a time-based x-axis, with time/date axis labels. Variable control charts that sample based on batches will generally use a numeric-based x-axis, with numeric axis labels.

SPC Analysis Charts

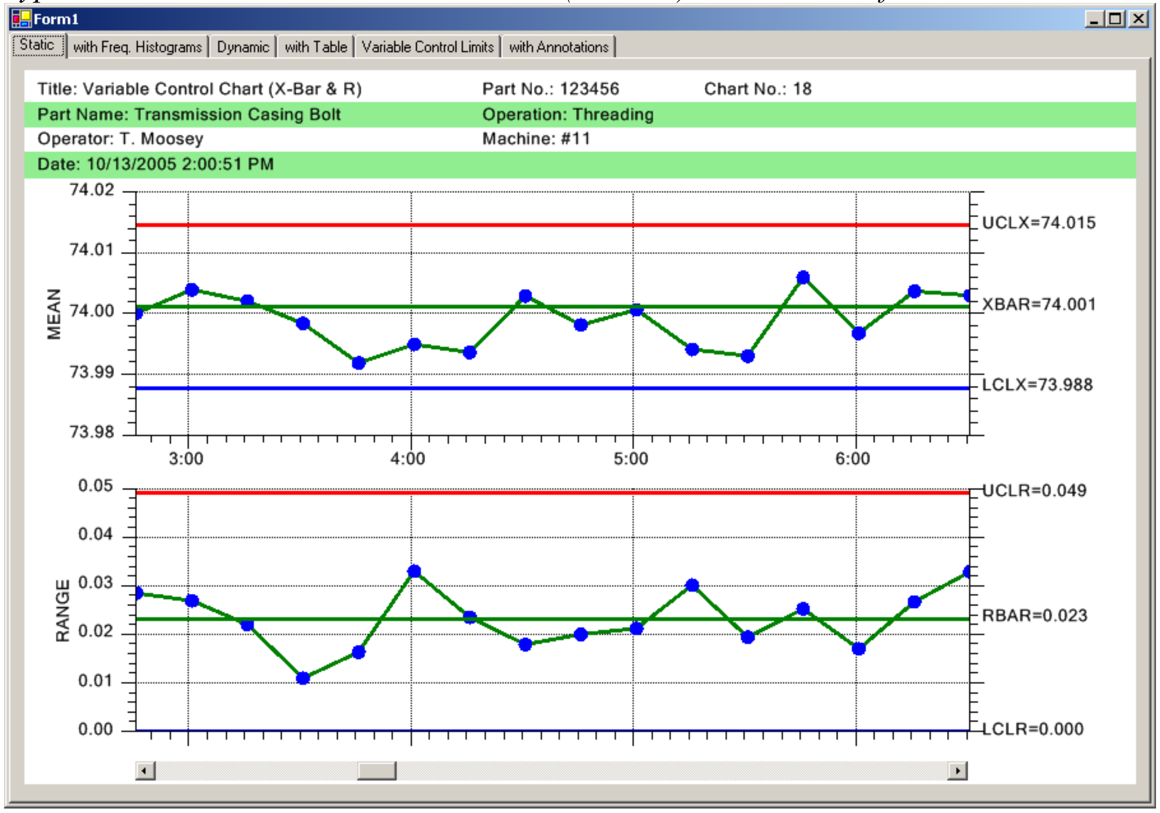
Quality engineers use other, specialized, charts in the analysis of SPC data. We have added chart classes that implement the following SPC analysis charts:

- Frequency Histograms
- Probability Charts
- Pareto Charts

Variable Control Charts

Variable Control Charts are for use with sampled quality data that can be assigned a specific numeric value, other than just 0 or 1. This might include, but is not limited to, the measurement of a critical dimension (height, length, width, radius, etc.), the weight a specific component, or the measurement of an important voltage. Common types of *Variable Control Charts* include X-Bar R (Mean and Range), X-Bar Sigma, Median and Range, X-R (Individual Range), EWMA, MA, MAMR (Moving Average/Moving Range), MAMS (Moving Average/Moving Sigma) and CuSum charts.

Typical Time-Base Variable Control Chart (X-Bar R) with header information



Title: Variable Control Chart (X-Bar & R) Part No.: 123456 Chart No.: 18
Part Name: Transmission Casing Bolt Operation: Threading
Operator: T. Moosey Machine: #11
Date: 10/13/2005 2:00:51 PM

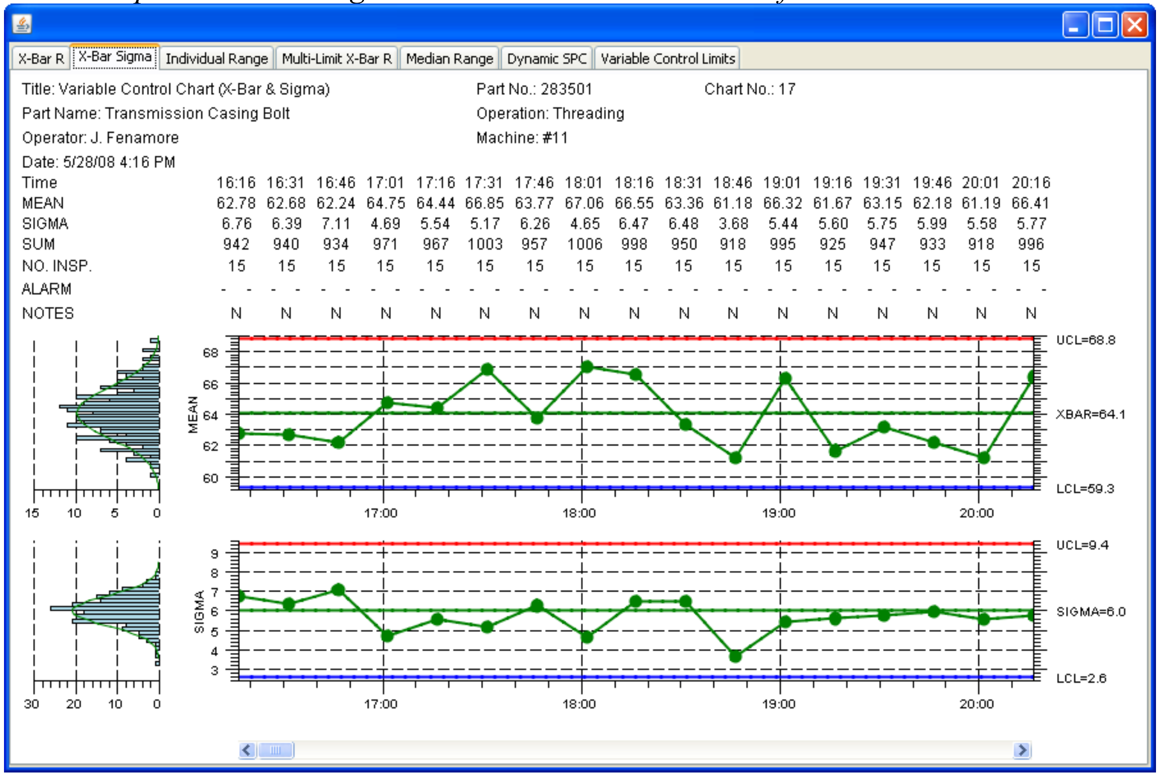
X-Bar R Chart – Also known as the Mean (or Average) and Range Chart

The X-Bar R chart monitors the trend of a critical process variable over time using a statistical sampling method that results in a subgroup of values at each subgroup interval. The X-Bar part of the chart plots the mean of each sample subgroup and the Range part of the chart monitors the difference between the minimum and maximum value in the subgroup.

X-Bar Sigma – Also known as the X-Bar S Chart

Very similar to the X-Bar R Chart, the X-Bar Sigma chart replaces the Range plot with a Sigma plot based on the standard deviation of the measured values within each subgroup. This is a more accurate way of establishing control limits if the sample size of the subgroup is moderately large (> 10). Though computationally more complicated, the use of a computer makes this a non-issue.

Fixed sample size X-Bar Sigma Control chart with header information

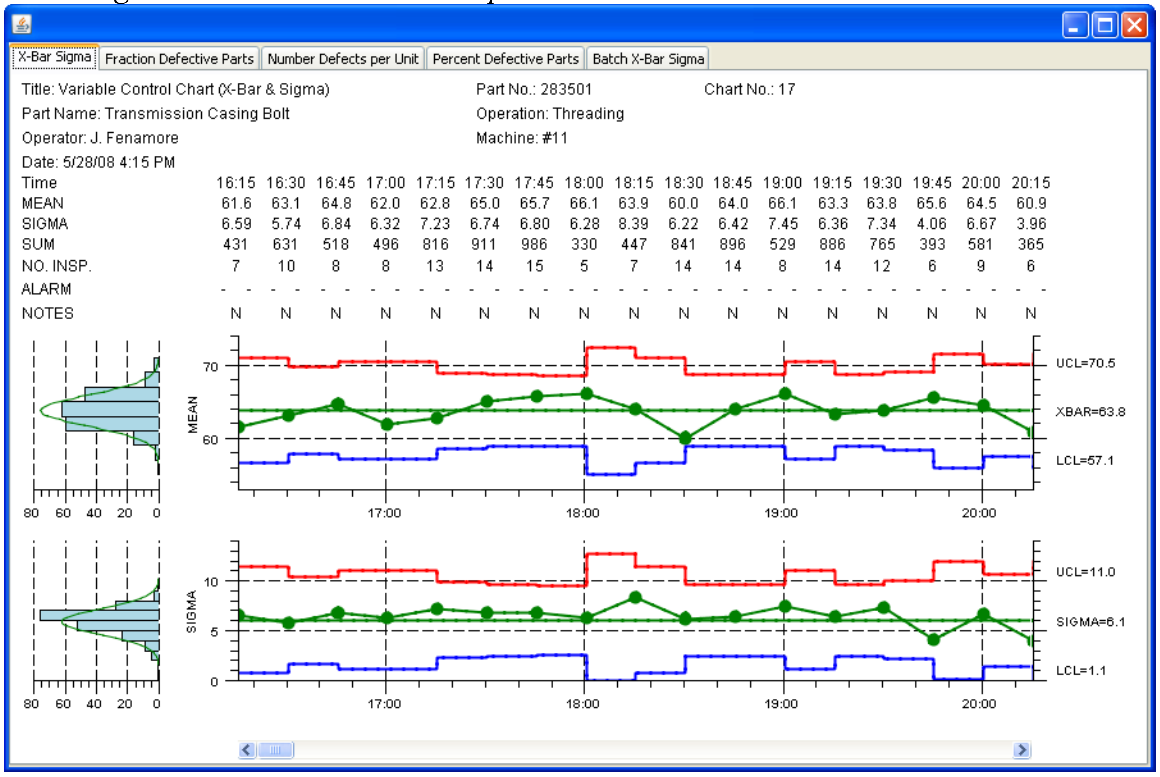


Title: Variable Control Chart (X-Bar & Sigma) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 5/28/08 4:16 PM

Time	16:16	16:31	16:46	17:01	17:16	17:31	17:46	18:01	18:16	18:31	18:46	19:01	19:16	19:31	19:46	20:01	20:16
MEAN	62.78	62.68	62.24	64.75	64.44	66.85	63.77	67.06	66.55	63.36	61.18	66.32	61.67	63.15	62.18	61.19	66.41
SIGMA	6.76	6.39	7.11	4.69	5.54	5.17	6.26	4.65	6.47	6.48	3.68	5.44	5.60	5.75	5.99	5.58	5.77
SUM	942	940	934	971	967	1003	957	1006	998	950	918	995	925	947	933	918	996
NO. INSP.	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The X-Bar Sigma chart can also be used if the sample subgroup size varies from sampling interval to sampling interval. In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are available. The X-Bar Sigma chart is the only variable control chart that can be used with a variable sample size.

X-Bar Sigma Chart with variable sample size



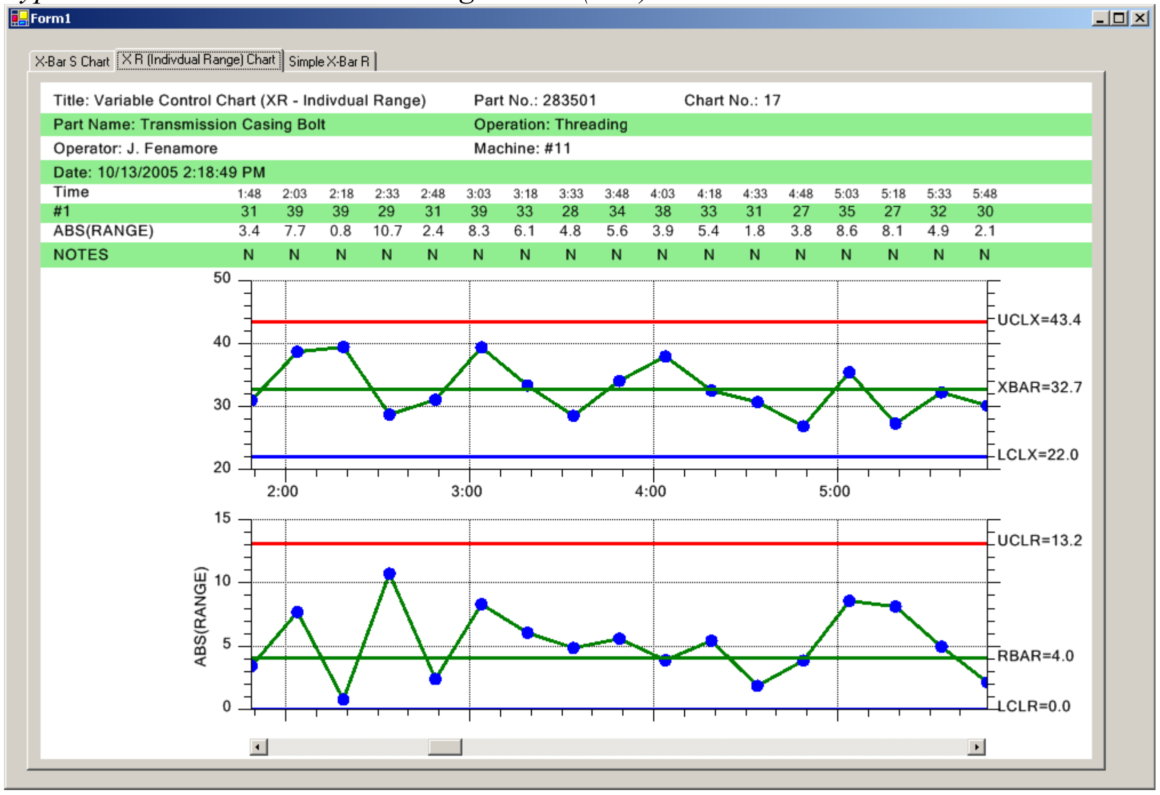
Title: Variable Control Chart (X-Bar & Sigma) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 5/28/08 4:15 PM

Time	16:15	16:30	16:45	17:00	17:15	17:30	17:45	18:00	18:15	18:30	18:45	19:00	19:15	19:30	19:45	20:00	20:15
MEAN	61.6	63.1	64.8	62.0	62.8	65.0	65.7	66.1	63.9	60.0	64.0	66.1	63.3	63.8	65.6	64.5	60.9
SIGMA	6.59	5.74	6.84	6.32	7.23	6.74	6.80	6.28	8.39	6.22	6.42	7.45	6.36	7.34	4.06	6.67	3.96
SUM	431	631	518	496	816	911	986	330	447	841	896	529	886	765	393	581	365
NO. INSP.	7	10	8	8	13	14	15	5	7	14	14	8	14	12	6	9	6
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Median Range – Also known as the Median and Range Chart

Very similar to the X-Bar R Chart, Median Range chart replaces the Mean plot with a Median plot representing the median of the measured values within each subgroup. The Median Range chart requires that the process be well behaved, where the variation in measured variables are (1) known to be distributed normally, (2) are not very often disturbed by assignable causes, and (3) are easily adjusted.

Typical Time-Based Individual Range Chart (X-R) with data table



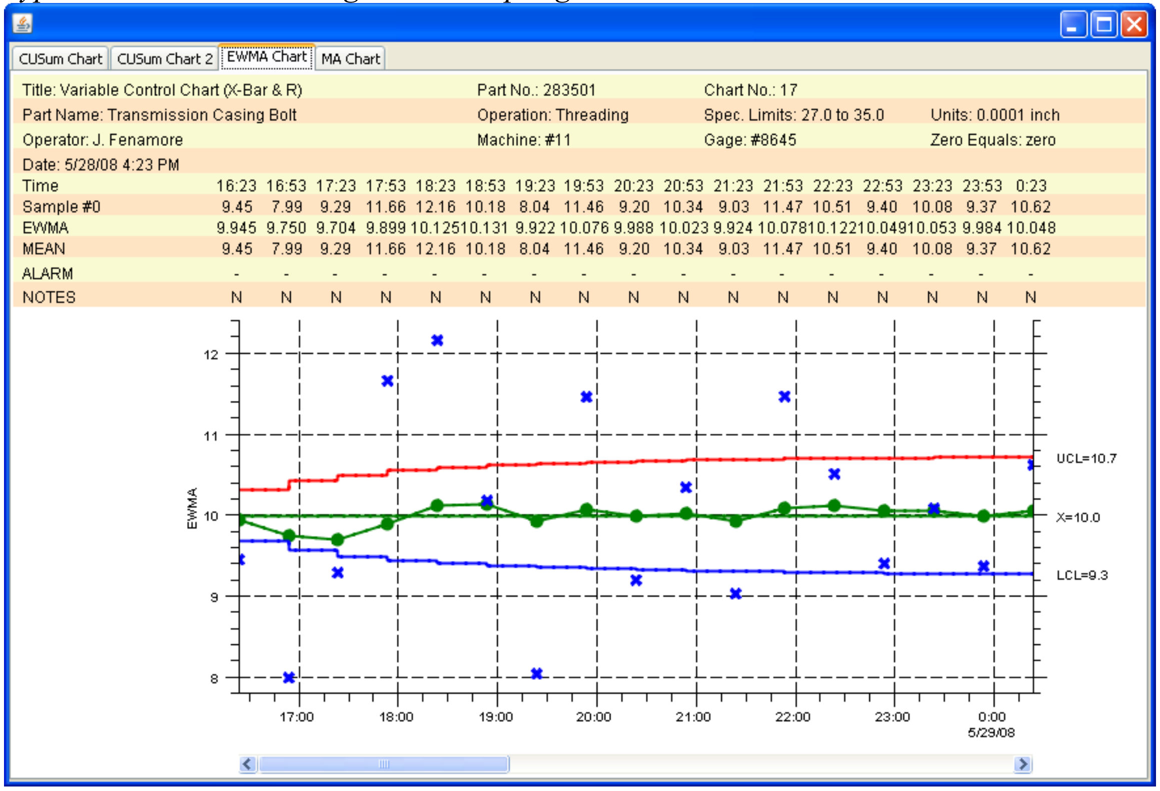
Title: Variable Control Chart (XR - Indivdual Range) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 10/13/2005 2:18:49 PM

Time	1:48	2:03	2:18	2:33	2:48	3:03	3:18	3:33	3:48	4:03	4:18	4:33	4:48	5:03	5:18	5:33	5:48
#1	31	39	39	29	31	39	33	28	34	38	33	31	27	35	27	32	30
ABS(RANGE)	3.4	7.7	0.8	10.7	2.4	8.3	6.1	4.8	5.6	3.9	5.4	1.8	3.8	8.6	8.1	4.9	2.1
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Individual Range Chart – Also known as the X-R Chart

The Individual Range Chart is used when the sample size for a subgroup is 1. This happens frequently when the inspection and collection of data for quality control purposes is automated and 100% of the units manufactured are analyzed. It also happens when the production rate is low and it is inconvenient to have sample sizes other than 1. The X part of the control chart plots the actual sampled value (not a mean or median) for each unit and the R part of the control chart plots a moving range, calculated using the current value of sampled value minus the previous value.

Typical EWMA Chart using Batch Sampling



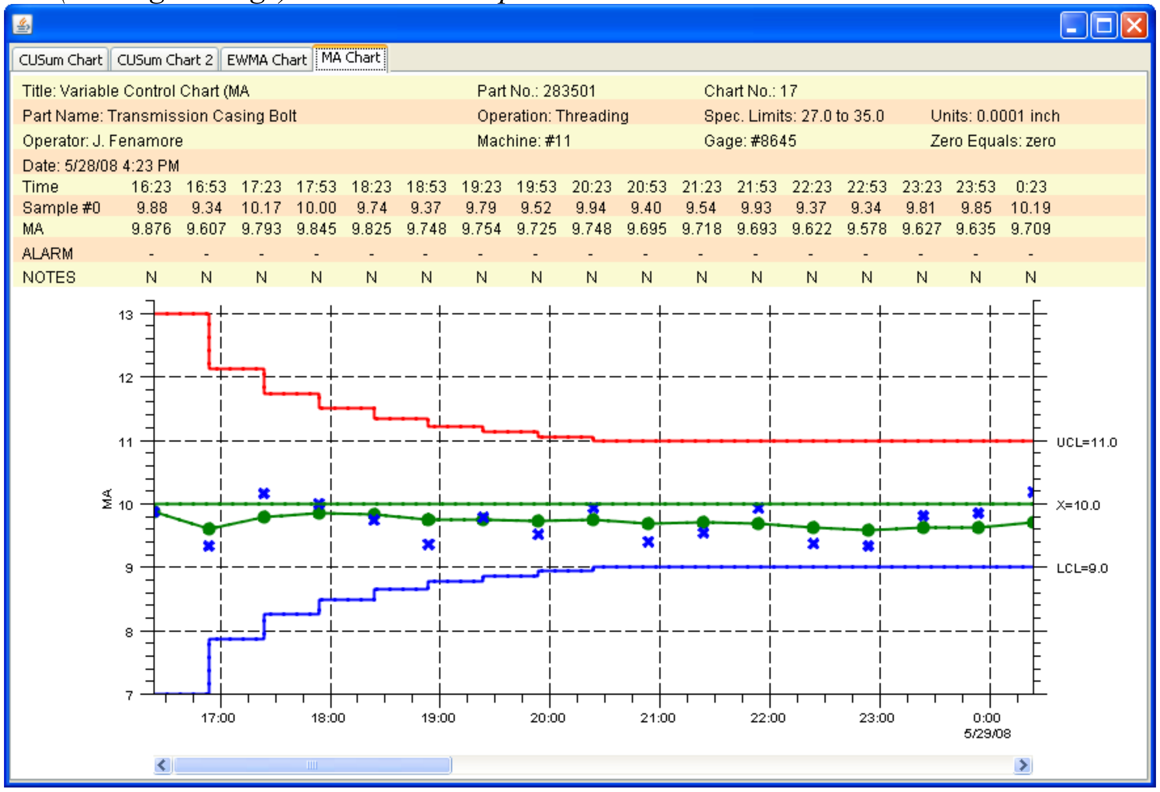
Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation: Threading	Spec. Limits: 27.0 to 35.0	Units: 0.0001 inch
Operator: J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 5/28/08 4:23 PM			

Time	16:23	16:53	17:23	17:53	18:23	18:53	19:23	19:53	20:23	20:53	21:23	21:53	22:23	22:53	23:23	23:53	0:23
Sample #0	9.45	7.99	9.29	11.66	12.16	10.18	8.04	11.46	9.20	10.34	9.03	11.47	10.51	9.40	10.08	9.37	10.62
EWMA	9.945	9.750	9.704	9.899	10.125	10.131	9.922	10.076	9.988	10.023	9.924	10.078	10.122	10.049	10.053	9.984	10.048
MEAN	9.45	7.99	9.29	11.66	12.16	10.18	8.04	11.46	9.20	10.34	9.03	11.47	10.51	9.40	10.08	9.37	10.62
ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

EWMA Chart – Exponentially Weighted Moving Average

The EWMA chart is an alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a weighted moving average of previous values to “smooth” the incoming data, minimizing the affect of random noise on the process. It weights the current and most recent values more heavily than older values, allowing the control line to react faster than a simple MA (Moving Average) plot to changes in the process. Like the Shewhart charts, if the EWMA value exceeds the calculated control limits, the process is considered out of control. While it is usually used where the process uses 100% inspection and the sample subgroup size is 1 (same is the I-R chart), it can also be used when sample subgroup sizes are greater than one.

MA (Moving Average) Chart with Sample Values Plotted



Title: Variable Control Chart (MA — Part No.: 283501 — Chart No.: 17

Part Name: Transmission Casing Bolt — Operation: Threading — Spec. Limits: 27.0 to 35.0 — Units: 0.0001 inch

Operator: J. Fenamore — Machine: #11 — Gage: #8645 — Zero Equals: zero

Date: 5/28/08 4:23 PM

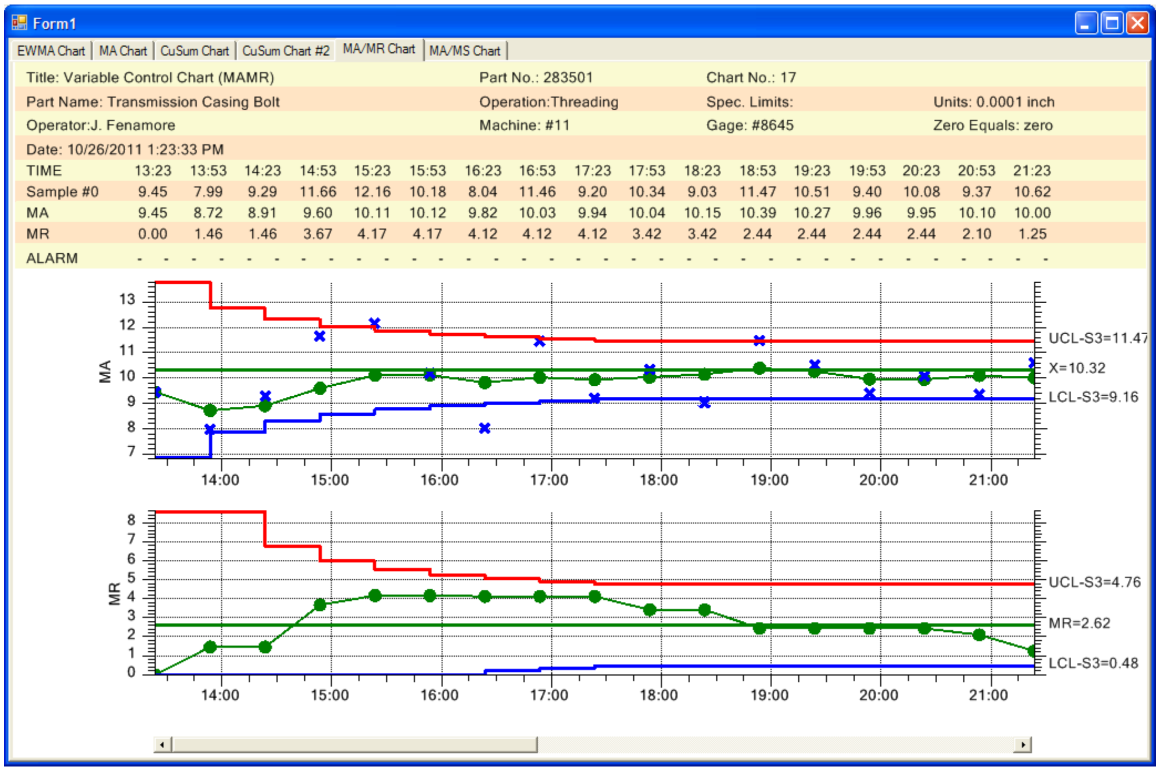
Time	16:23	16:53	17:23	17:53	18:23	18:53	19:23	19:53	20:23	20:53	21:23	21:53	22:23	22:53	23:23	23:53	0:23
Sample #0	9.88	9.34	10.17	10.00	9.74	9.37	9.79	9.52	9.94	9.40	9.54	9.93	9.37	9.34	9.81	9.85	10.19
MA	9.876	9.607	9.793	9.845	9.825	9.748	9.754	9.725	9.748	9.695	9.718	9.693	9.622	9.578	9.627	9.635	9.709
ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

MA Chart – Moving Average

The MA chart is another alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a moving average, where the previous (N-1) sample values of the process variable are averaged together along with the process value to produce the current chart value. This helps to “smooth” the incoming data, minimizing the affect of random noise on the process. Unlike the EWMA chart, the MA chart weights the current and previous (N-1) values equally in the average. While the MA chart can often detect small changes in the process mean faster than the Shewhart chart types, it is generally considered inferior to the EWMA chart. Like the Shewhart charts, if the MA value exceeds the calculated control limits, the process is considered out of control.

MAMR Chart – Moving Average / Moving Range

MAMR (Moving Average/Moving Range) Chart with Sample Values Plotted



Title: Variable Control Chart (MAMR) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation:Threading Spec. Limits: Units: 0.0001 inch

Operator:J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero

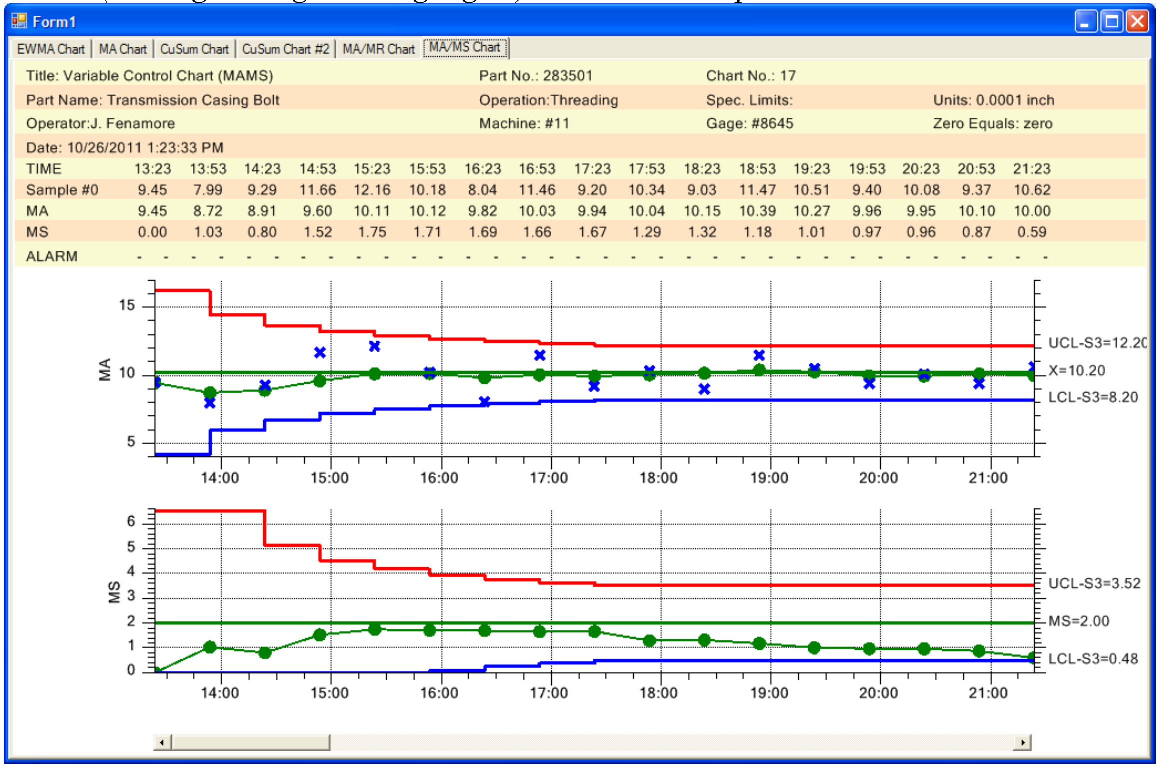
Date: 10/26/2011 1:23:33 PM

TIME	13:23	13:53	14:23	14:53	15:23	15:53	16:23	16:53	17:23	17:53	18:23	18:53	19:23	19:53	20:23	20:53	21:23
Sample #0	9.45	7.99	9.29	11.66	12.16	10.18	8.04	11.46	9.20	10.34	9.03	11.47	10.51	9.40	10.08	9.37	10.62
MA	9.45	8.72	8.91	9.60	10.11	10.12	9.82	10.03	9.94	10.04	10.15	10.39	10.27	9.96	9.95	10.10	10.00
MR	0.00	1.46	1.46	3.67	4.17	4.17	4.12	4.12	4.12	3.42	3.42	2.44	2.44	2.44	2.44	2.10	1.25
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -

The MAMR chart combines our Moving Average chart with a Moving Range chart. The Moving Average chart is primary (topmost) chart, and the Moving Range chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Range, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving range statistics.

MAMS Chart – Moving Average / Moving Sigma

MAMS (Moving Average/Moving Sigma) Chart with Sample Values Plotted



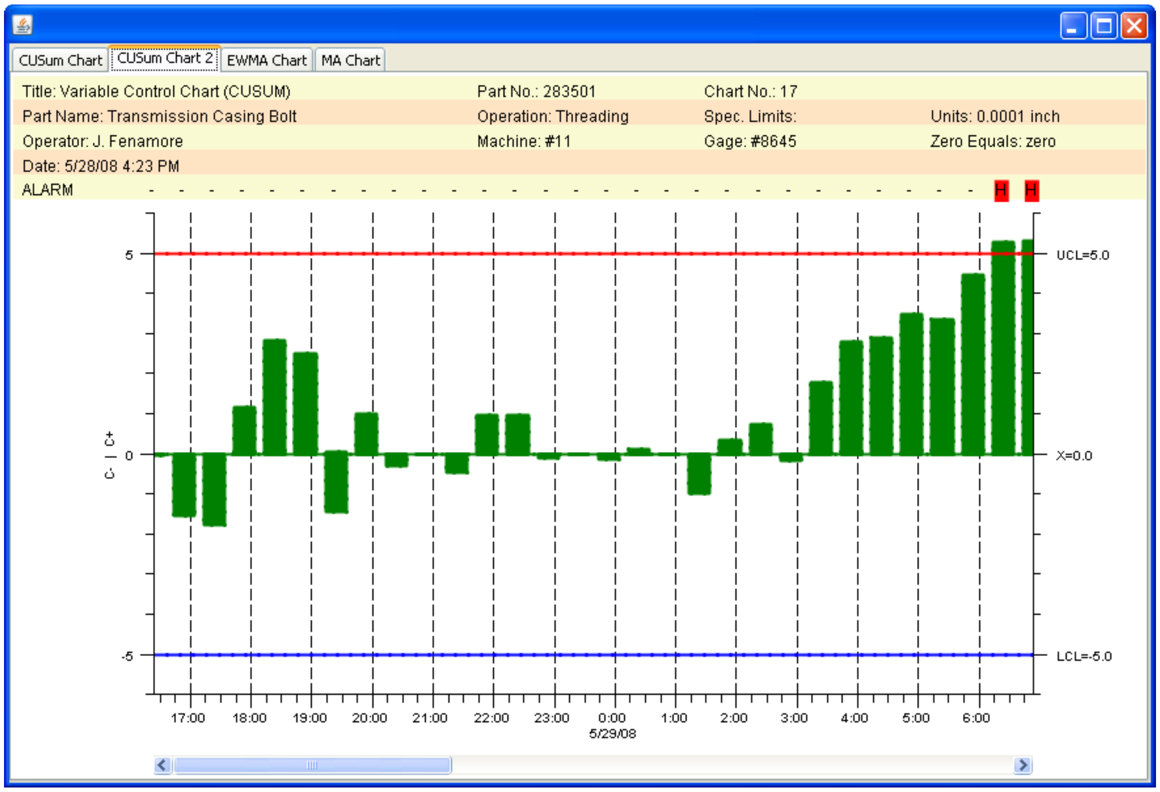
Form1
EWMA Chart | MA Chart | CuSum Chart | CuSum Chart #2 | MA/MR Chart | MA/MS Chart

Title: Variable Control Chart (MAMS) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation:Threading Spec. Limits: Units: 0.0001 inch
Operator:J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero
Date: 10/26/2011 1:23:33 PM

TIME	13:23	13:53	14:23	14:53	15:23	15:53	16:23	16:53	17:23	17:53	18:23	18:53	19:23	19:53	20:23	20:53	21:23
Sample #0	9.45	7.99	9.29	11.66	12.16	10.18	8.04	11.46	9.20	10.34	9.03	11.47	10.51	9.40	10.08	9.37	10.62
MA	9.45	8.72	8.91	9.60	10.11	10.12	9.82	10.03	9.94	10.04	10.15	10.39	10.27	9.96	9.95	10.10	10.00
MS	0.00	1.03	0.80	1.52	1.75	1.71	1.69	1.66	1.67	1.29	1.32	1.18	1.01	0.97	0.96	0.87	0.59
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -

The MAMS chart combines our Moving Average chart with a Moving Sigma chart. The Moving Average chart is primary (topmost) chart, and the Moving Sigma chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Sigma, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving sigma statistics.

Tabular CuSum Chart



Title: Variable Control Chart (CUSUM) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch
Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero
Date: 5/28/08 4:23 PM

CuSum Chart – Tabular, one-sided, upper and lower cumulative sum

The CuSum chart is a specialized control chart, which like the EWMA and MA charts, is considered to be more efficient that the Shewhart charts at detecting small shifts in the process mean, particularly if the mean shift is less than 2 sigma. There are several types of CuSum charts, but the easiest to use and the most accurate is considered the tabular CuSum chart and that is the one implemented in this software. The tabular cusum works by accumulating deviations that are above the process mean in one statistic (C+) and accumulating deviations below the process mean in a second statistic (C-). If either statistic (C+ or C-) falls outside of the calculated limits, the process is considered out of control.

Measured Data and Calculated Value Tables

Standard worksheets used to gather and plot SPC data consist of three main parts.

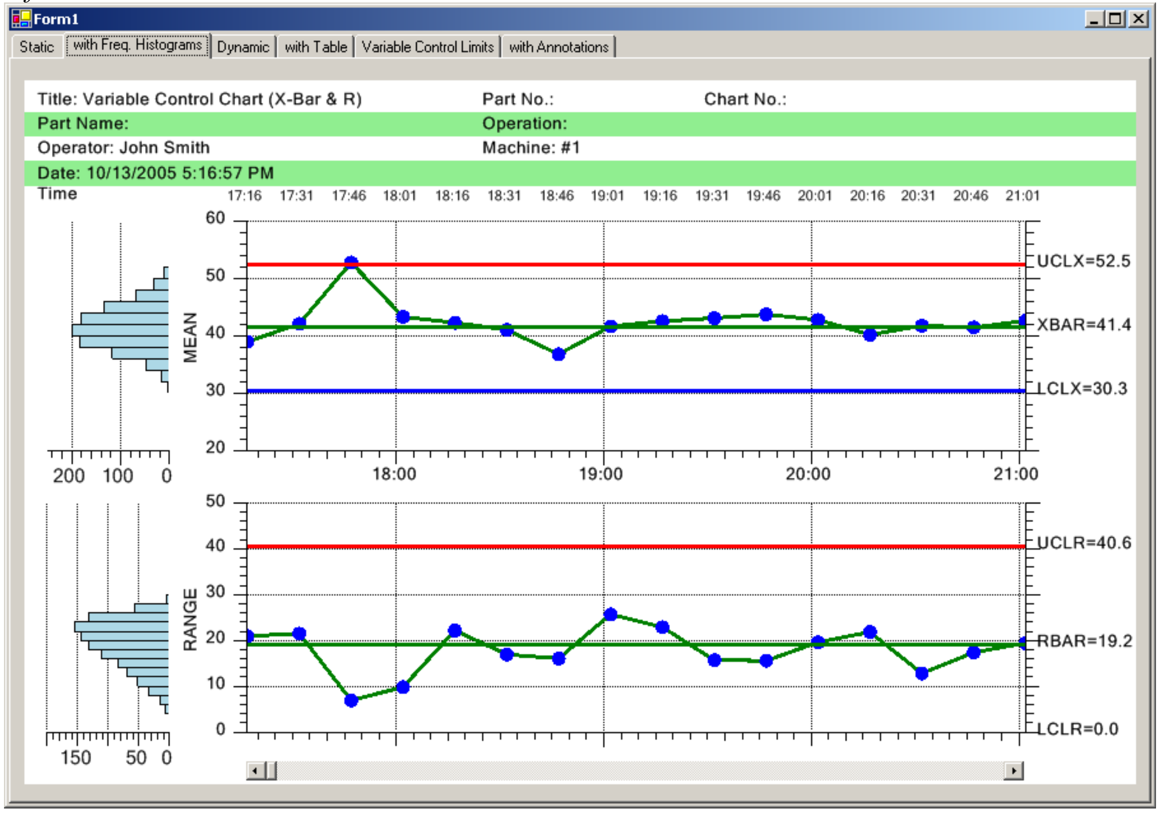
⑤ The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.

⑤ The second part is the measurement data recording and calculation section, organized as a table, recording the sampled and calculated data in a neat, readable fashion.

⑤ The third part, the actual SPC chart, plots the calculated SPC values for the sample group

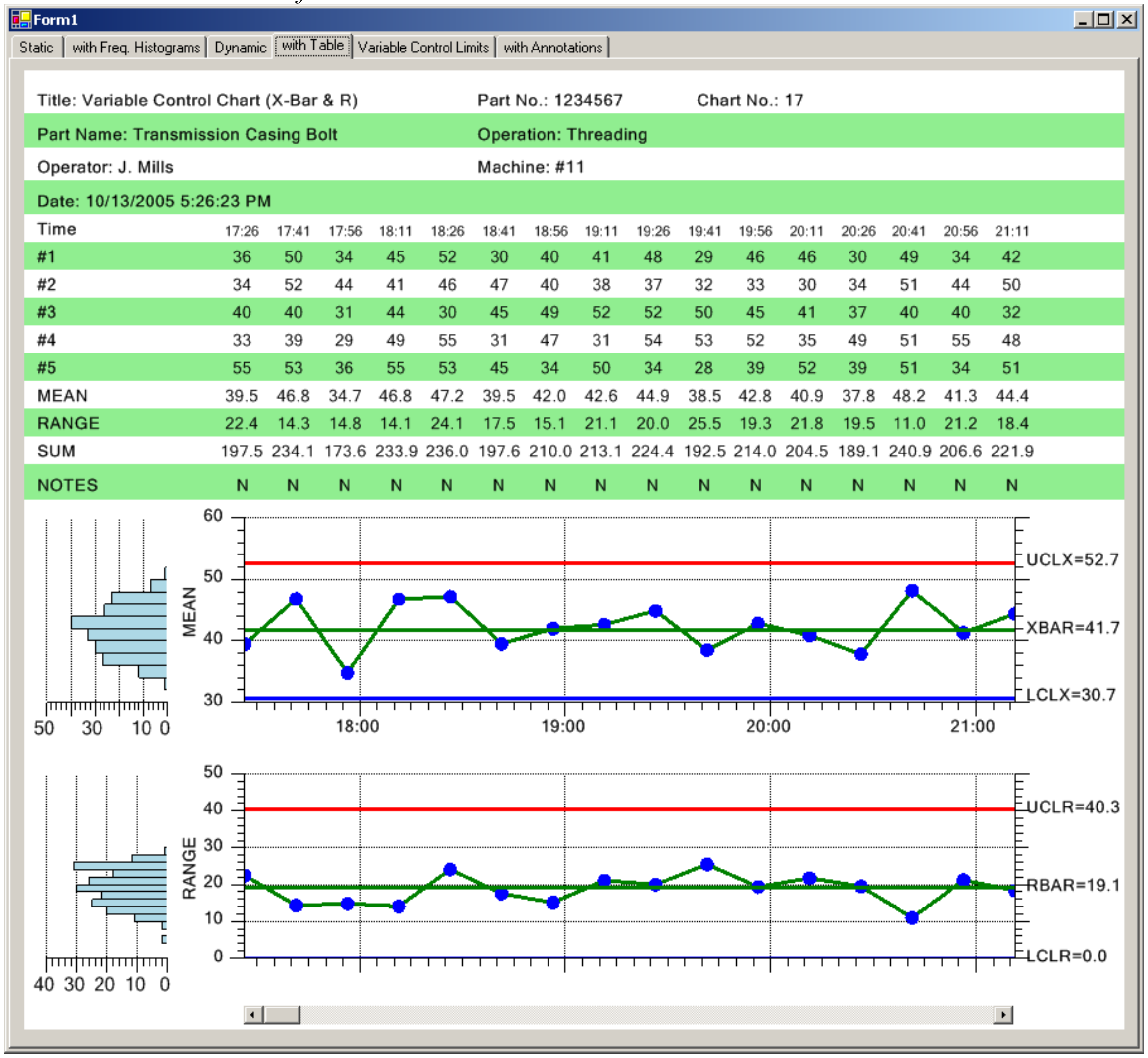
The *Variable Control Chart* templates that we have created have options that enable the programmer to customize and automatically include header information along with a table of the measurement and calculated data, in the SPC chart. Enable the scrollbar option and you can display the tabular measurement data and SPC plots for a window of 8-20 subgroups, from a much larger collection of measurement data represented hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.

Scrollable Time-Based XBar-R Chart with frequency histograms and basic header information



Title: Variable Control Chart (X-Bar & R) Part No.: Chart No.:
Part Name: Operation:
Operator: John Smith Machine: #1
Date: 10/13/2005 5:16:57 PM

Scrollable Time-Based XBar-R Chart with frequency histograms, header, measurement and calculated value information



Title: Variable Control Chart (X-Bar & R) Part No.: 1234567 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Mills Machine: #11

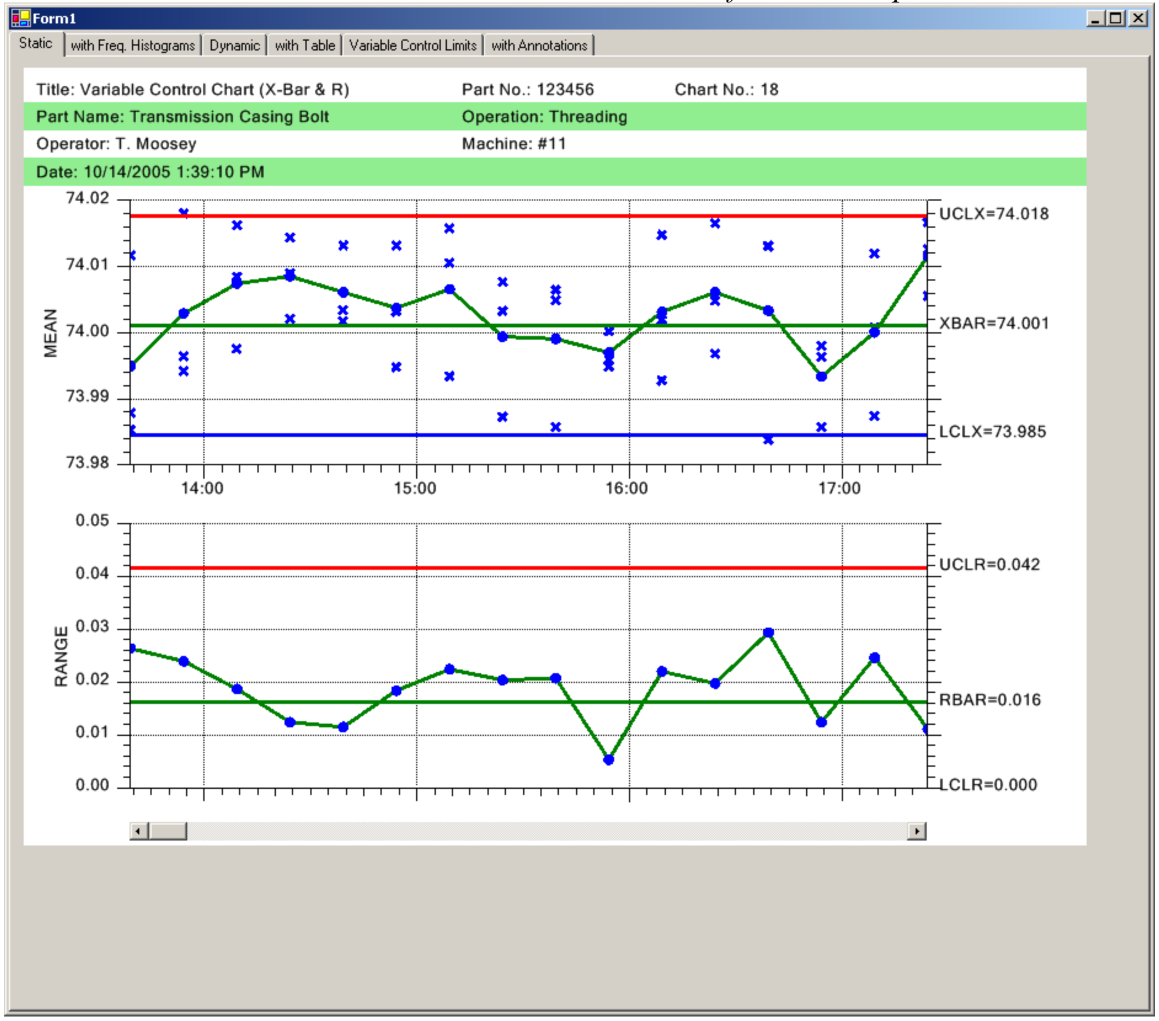
Date: 10/13/2005 5:26:23 PM

Time	17:26	17:41	17:56	18:11	18:26	18:41	18:56	19:11	19:26	19:41	19:56	20:11	20:26	20:41	20:56	21:11
#1	36	50	34	45	52	30	40	41	48	29	46	46	30	49	34	42
#2	34	52	44	41	46	47	40	38	37	32	33	30	34	51	44	50
#3	40	40	31	44	30	45	49	52	52	50	45	41	37	40	40	32
#4	33	39	29	49	55	31	47	31	54	53	52	35	49	51	55	48
#5	55	53	36	55	53	45	34	50	34	28	39	52	39	51	34	51
MEAN	39.5	46.8	34.7	46.8	47.2	39.5	42.0	42.6	44.9	38.5	42.8	40.9	37.8	48.2	41.3	44.4
RANGE	22.4	14.3	14.8	14.1	24.1	17.5	15.1	21.1	20.0	25.5	19.3	21.8	19.5	11.0	21.2	18.4
SUM	197.5	234.1	173.6	233.9	236.0	197.6	210.0	213.1	224.4	192.5	214.0	204.5	189.1	240.9	206.6	221.9
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Scatter Plots of the Actual Sampled Data

In some cases it useful to plot the actual values of a sample subgroup along with the sample subgroup mean or median. Plot these samples in the SPC chart using additional scatter plots.

Scrollable Time-Based XBar-R Chart with Scatter Plot of Actual Sampled Data

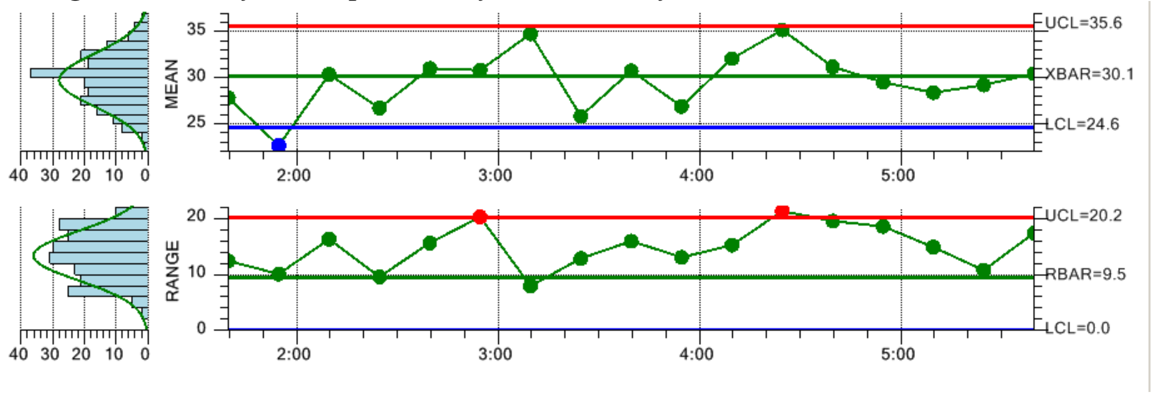


Title: Variable Control Chart (X-Bar & R) Part No.: 123456 Chart No.: 18
Part Name: Transmission Casing Bolt Operation: Threading
Operator: T. Moosey Machine: #11
Date: 10/14/2005 1:39:10 PM

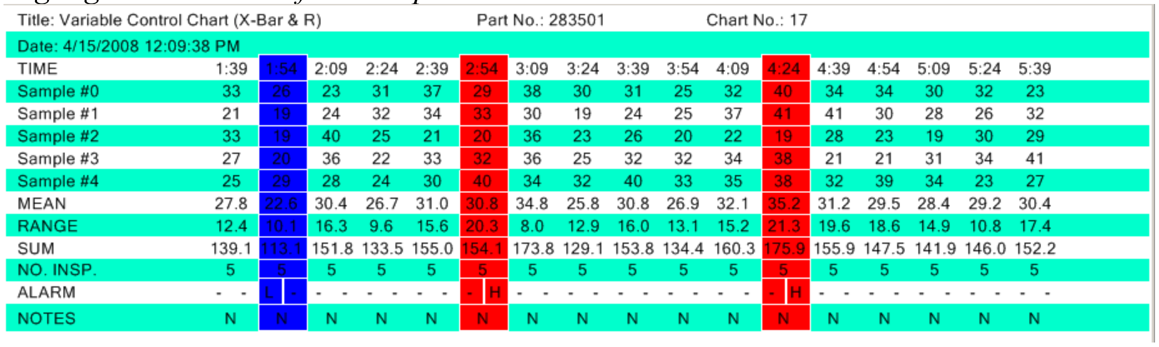
Alarm Notification

Typically, when a process value exceeds a control limit, an alarm condition exists. In order to make sure that the program user identifies an alarm you can emphasize the alarm in several different ways. You can trap the alarm condition using an event delegate, log the alarm to the notes log, highlight the data point symbol in the chart where the alarm occurs, display an alarm status line in the data table, or highlight the entire column of the sample interval where the alarm occurs.

Change the color of a data point that falls outside of alarm limits.



Highlight the column of the sample interval where the alarm occurs

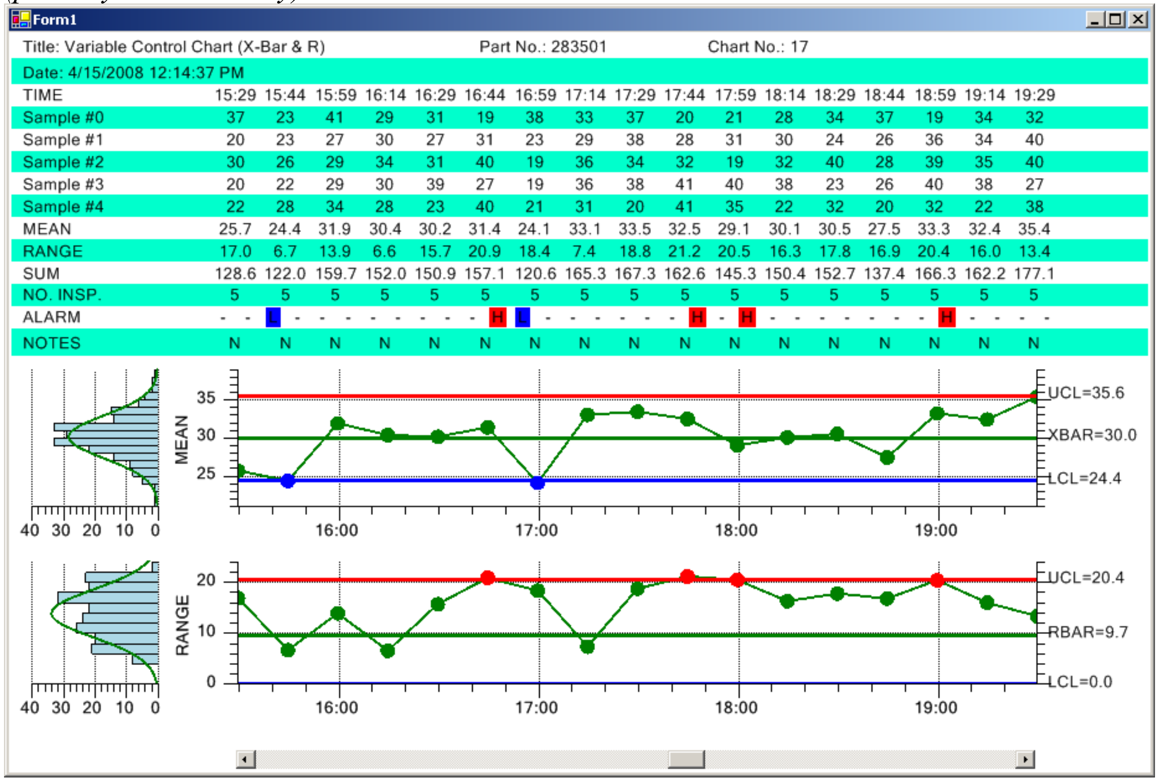


Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Date: 4/15/2008 12:09:38 PM

TIME	1:39	1:54	2:09	2:24	2:39	2:54	3:09	3:24	3:39	3:54	4:09	4:24	4:39	4:54	5:09	5:24	5:39
Sample #0	33	26	23	31	37	29	38	30	31	25	32	40	34	34	30	32	23
Sample #1	21	19	24	32	34	33	30	19	24	25	37	41	41	30	28	26	32
Sample #2	33	19	40	25	21	20	36	23	26	20	22	19	28	23	19	30	29
Sample #3	27	20	36	22	33	32	36	25	32	32	34	38	21	21	31	34	41
Sample #4	25	29	28	24	30	40	34	32	40	33	35	38	32	39	34	23	27
MEAN	27.8	22.6	30.4	26.7	31.0	30.8	34.8	25.8	30.8	26.9	32.1	35.2	31.2	29.5	28.4	29.2	30.4
RANGE	12.4	10.1	16.3	9.6	15.6	20.3	8.0	12.9	16.0	13.1	15.2	21.3	19.6	18.6	14.9	10.8	17.4
SUM	139.1	113.1	151.8	133.5	155.0	154.1	173.8	129.1	153.8	134.4	160.3	175.9	155.9	147.5	141.9	146.0	152.2
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
ALARM	- -	L -	- -	- -	- -	- H	- -	- -	- -	- -	- -	- H	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

An alarm status line highlights an alarm condition, and lets you know when chart the (primary or secondary) the alarm occurs in.



Form1

Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Date: 4/15/2008 12:14:37 PM

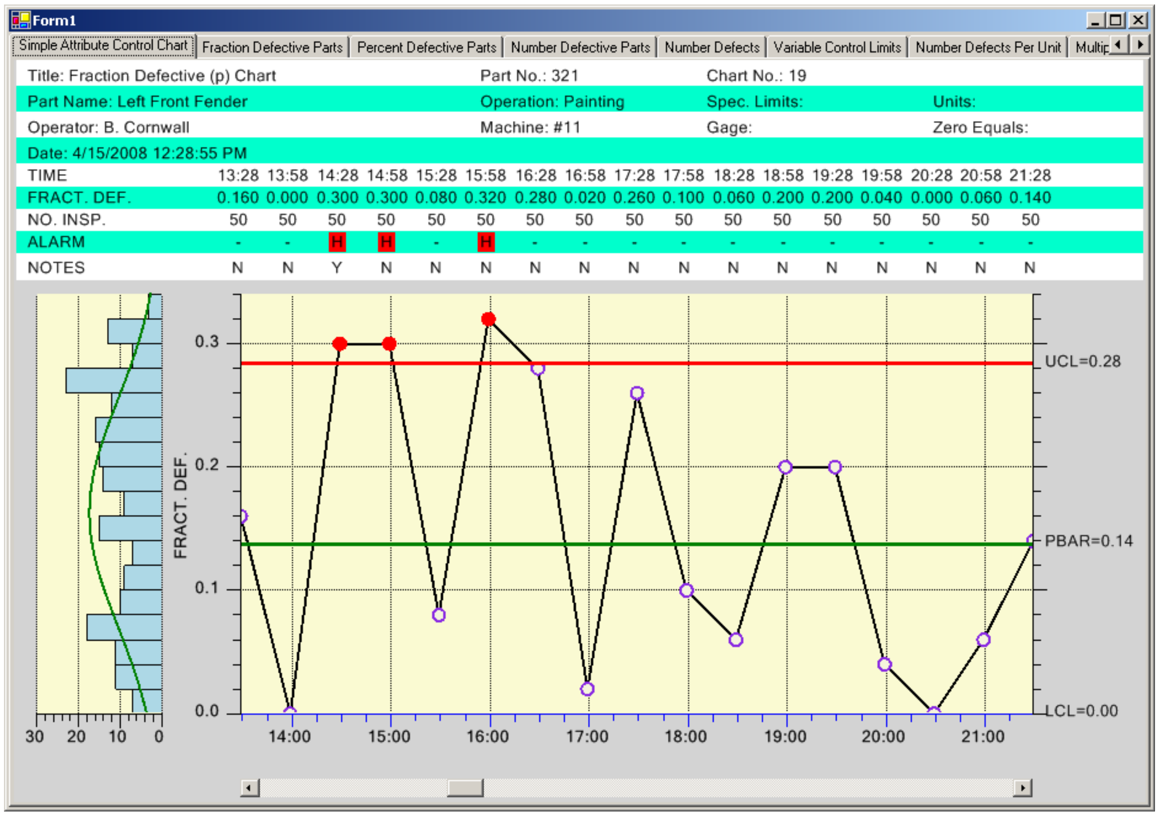
TIME	15:29	15:44	15:59	16:14	16:29	16:44	16:59	17:14	17:29	17:44	17:59	18:14	18:29	18:44	18:59	19:14	19:29
Sample #0	37	23	41	29	31	19	38	33	37	20	21	28	34	37	19	34	32
Sample #1	20	23	27	30	27	31	23	29	38	28	31	30	24	26	36	34	40
Sample #2	30	26	29	34	31	40	19	36	34	32	19	32	40	28	39	35	40
Sample #3	20	22	29	30	39	27	19	36	38	41	40	38	23	26	40	38	27
Sample #4	22	28	34	28	23	40	21	31	20	41	35	22	32	20	32	22	38
MEAN	25.7	24.4	31.9	30.4	30.2	31.4	24.1	33.1	33.5	32.5	29.1	30.1	30.5	27.5	33.3	32.4	35.4
RANGE	17.0	6.7	13.9	6.6	15.7	20.9	18.4	7.4	18.8	21.2	20.5	16.3	17.8	16.9	20.4	16.0	13.4
SUM	128.6	122.0	159.7	152.0	150.9	157.1	120.6	165.3	167.3	162.6	145.3	150.4	152.7	137.4	166.3	162.2	177.1
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
ALARM	- -	L -	- -	- -	- -	- H	L -	- -	- -	- H	- H	- -	- -	- -	- H	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

These alarm highlight features apply to both variable control and attribute control charts.

Attribute Control Charts

Attribute Control Charts are a set of control charts specifically designed for tracking defects (also called non-conformities). These types of defects are binary in nature (yes/no), where a part has one or more defects, or it doesn't. Examples of defects are paint scratches, discolorations, breaks in the weave of a textile, dents, cuts, etc. Think of the last car that you bought. The defects in each sample group are counted and run through some statistical calculations. Depending on the type of *Attribute Control Chart*, the number of defective parts are tracked (p-chart and np-chart) , or alternatively, the number of defects are tracked (u-chart, c-chart). The difference in terminology "number of defective parts" and "number of defects" is highly significant, since a single part not only can have multiple defect categories (scratch, color, dent, etc), it can also have multiple defects per category. A single part may have 0 – N defects. So keeping track of the number of defective parts is statistically different from keeping track of the number of defects. This affects the way the control limits for each chart are calculated.

Typical Time-Based Attribute Control Chart (p-Chart)



Title: Fraction Defective (p) Chart | Part No.: 321 | Chart No.: 19
Part Name: Left Front Fender | Operation: Painting | Spec. Limits: | Units:
Operator: B. Cornwall | Machine: #11 | Gage: | Zero Equals:
Date: 4/15/2008 12:28:55 PM

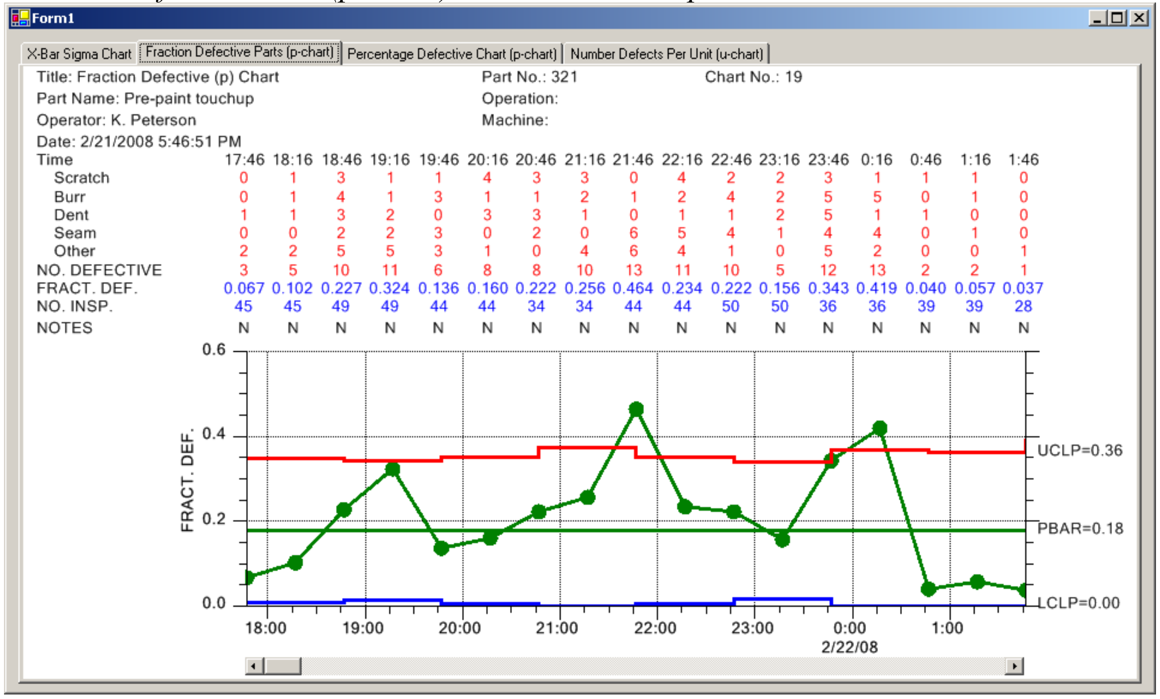
TIME	13:28	13:58	14:28	14:58	15:28	15:58	16:28	16:58	17:28	17:58	18:28	18:58	19:28	19:58	20:28	20:58	21:28
FRACT. DEF.	0.160	0.000	0.300	0.300	0.080	0.320	0.280	0.020	0.260	0.100	0.060	0.200	0.200	0.040	0.000	0.060	0.140
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	H	H	-	H	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N

p-Chart - Also known as the Percent or Fraction Defective Parts Chart

For a sample subgroup, the number of defective parts is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

The p-Chart chart can also be used if the sample subgroup size varies from sampling interval to sampling interval. In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are available. Both the *Fraction Defective Parts and Percent Defective Parts* control charts come in versions that support variable sample sized for a subgroup.

Fraction Defective Parts (p-Chart) with variable sample size



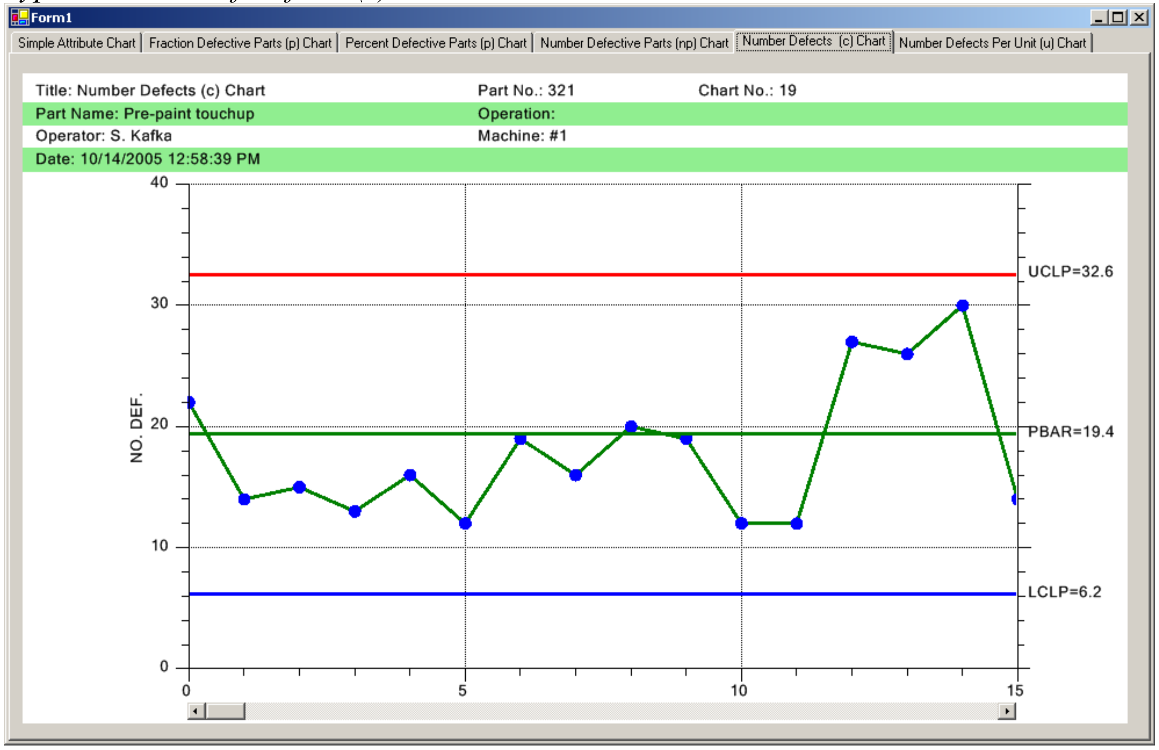
Title: Fraction Defective (p) Chart — Part No.: 321 — Chart No.: 19
Part Name: Pre-paint touchup — Operation:
Operator: K. Peterson — Machine:
Date: 2/21/2008 5:46:51 PM

Time	17:46	18:16	18:46	19:16	19:46	20:16	20:46	21:16	21:46	22:16	22:46	23:16	23:46	0:16	0:46	1:16	1:46
Scratch	0	1	3	1	1	4	3	3	0	4	2	2	3	1	1	1	0
Burr	0	1	4	1	3	1	1	2	1	2	4	2	5	5	0	1	0
Dent	1	1	3	2	0	3	3	1	0	1	1	2	5	1	1	0	0
Seam	0	0	2	2	3	0	2	0	6	5	4	1	4	4	0	1	0
Other	2	2	5	5	3	1	0	4	6	4	1	0	5	2	0	0	1
NO. DEFECTIVE	3	5	10	11	6	8	8	10	13	11	10	5	12	13	2	2	1
FRACT. DEF.	0.067	0.102	0.227	0.324	0.136	0.160	0.222	0.256	0.464	0.234	0.222	0.156	0.343	0.419	0.040	0.057	0.037
NO. INSP.	45	45	49	49	44	44	34	34	44	44	50	50	36	36	39	39	28
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

np-Chart – Also known as the Number Defective Parts Chart

For a sample subgroup, the number of defective parts is measured and plotted as a simple count. Statistically, in order to compare number of defective parts for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

Typical Number of Defects (c)


Form1
Simple Attribute Chart
Fraction Defective Parts (p) Chart
Percent Defective Parts (p) Chart
Number Defective Parts (np) Chart
Number Defects (c) Chart
Number Defects Per Unit (u) Chart

Title: Number Defects (c) Chart Part No.: 321 Chart No.: 19
Part Name: Pre-paint touchup Operation:
Operator: S. Kafka Machine: #1
Date: 10/14/2005 12:58:39 PM

NO. DEF.
40
30
20
10
0
0
5
10
15
UCLP=32.6
PBAR=19.4
LCLP=6.2

c-Chart - Also known as the Number of Defects or Number of Non-Conformities Chart

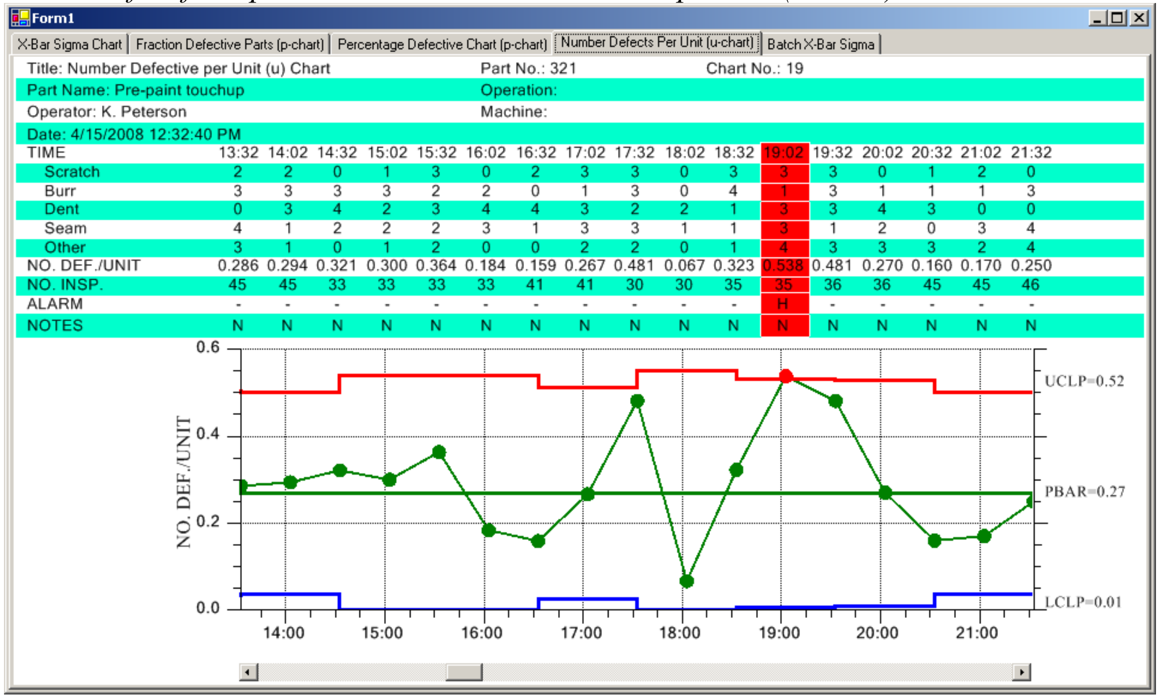
For a sample subgroup, the number of times a defect occurs is measured and plotted as a simple count. Statistically, in order to compare number of defects for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

u-Chart – Also known as the Number of Defects per Unit or Number of Non-Conformities per Unit Chart

For a sample subgroup, the number of times a defect occurs is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

The u-Chart chart can also be used if the sample subgroup size varies from sampling interval to sampling interval. In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are available.

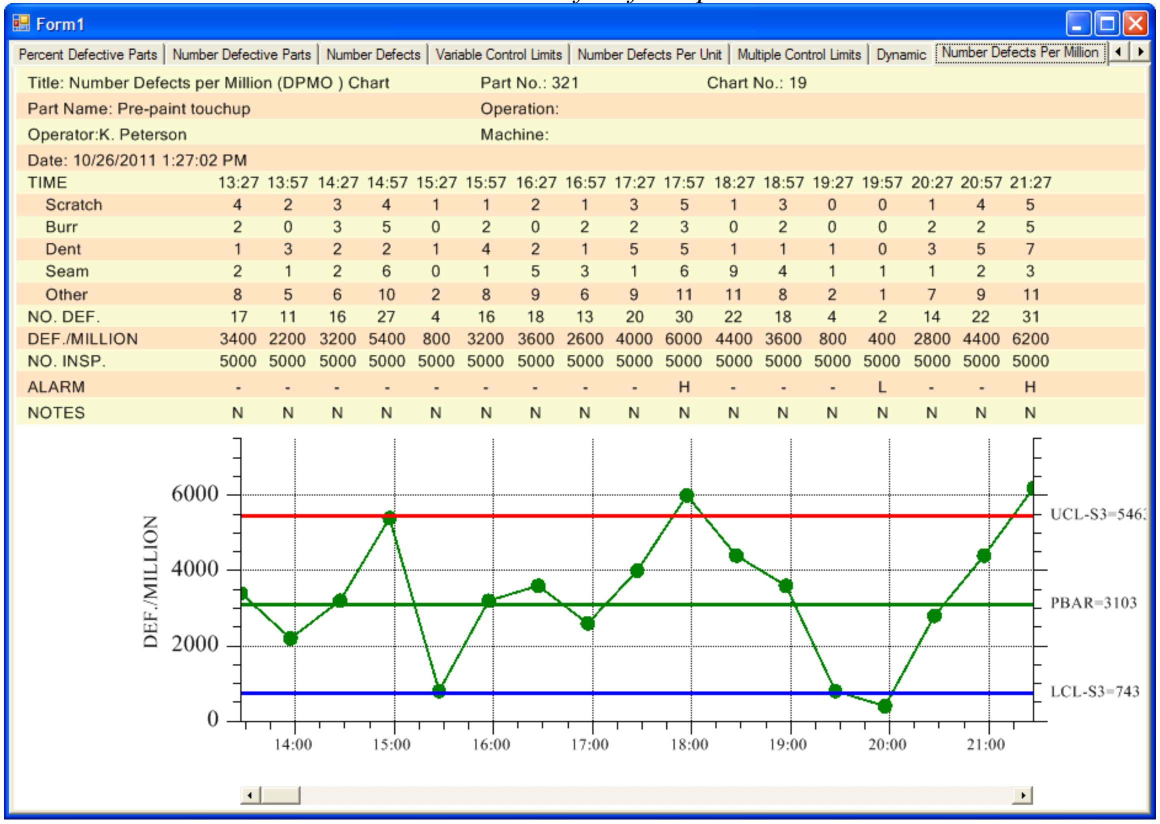
Number of Defects per Unit Chart with variable sample size (u-chart)



Title: Number Defective per Unit (u) Chart   Part No.: 321   Chart No.: 19
Part Name: Pre-paint touchup   Operation:
Operator: K. Peterson   Machine:
Date: 4/15/2008 12:32:40 PM

TIME	13:32	14:02	14:32	15:02	15:32	16:02	16:32	17:02	17:32	18:02	18:32	19:02	19:32	20:02	20:32	21:02	21:32
Scratch	2	2	0	1	3	0	2	3	3	0	3	3	3	0	1	2	0
Burr	3	3	3	3	2	2	0	1	3	0	4	1	3	1	1	1	3
Dent	0	3	4	2	3	4	4	3	2	2	1	3	3	4	3	0	0
Seam	4	1	2	2	2	3	1	3	3	1	1	3	1	2	0	3	4
Other	3	1	0	1	2	0	0	2	2	0	1	4	3	3	3	2	4
NO. DEF./UNIT	0.286	0.294	0.321	0.300	0.364	0.184	0.159	0.267	0.481	0.067	0.323	0.538	0.481	0.270	0.160	0.170	0.250
NO. INSP.	45	45	33	33	33	33	41	41	30	30	35	35	36	36	45	45	46
ALARM	-	-	-	-	-	-	-	-	-	-	-	H	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

DPMO Chart – Also known as the Number of Defects per Million Chart



Title: Number Defects per Million (DPMO) Chart   Part No.: 321   Chart No.: 19
Part Name: Pre-paint touchup   Operation:
Operator:K. Peterson   Machine:
Date: 10/26/2011 1:27:02 PM

TIME	13:27	13:57	14:27	14:57	15:27	15:57	16:27	16:57	17:27	17:57	18:27	18:57	19:27	19:57	20:27	20:57	21:27
Scratch	4	2	3	4	1	1	2	1	3	5	1	3	0	0	1	4	5
Burr	2	0	3	5	0	2	0	2	2	3	0	2	0	0	2	2	5
Dent	1	3	2	2	1	4	2	1	5	5	1	1	1	0	3	5	7
Seam	2	1	2	6	0	1	5	3	1	6	9	4	1	1	1	2	3
Other	8	5	6	10	2	8	9	6	9	11	11	8	2	1	7	9	11
NO. DEF.	17	11	16	27	4	16	18	13	20	30	22	18	4	2	14	22	31
DEF./MILLION	3400	2200	3200	5400	800	3200	3600	2600	4000	6000	4400	3600	800	400	2800	4400	6200
NO. INSP.	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000
ALARM	-	-	-	-	-	-	-	-	-	H	-	-	-	L	-	-	H
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

This Attribute Control chart is a combination of the u-chart and the c-chart. The chart normalizes the defect rate, expressing it as defects per million. The chart displays the defect rate as defects per million. The table above gives the defect count in both absolute terms, and in the normalized defects per million used by the chart.

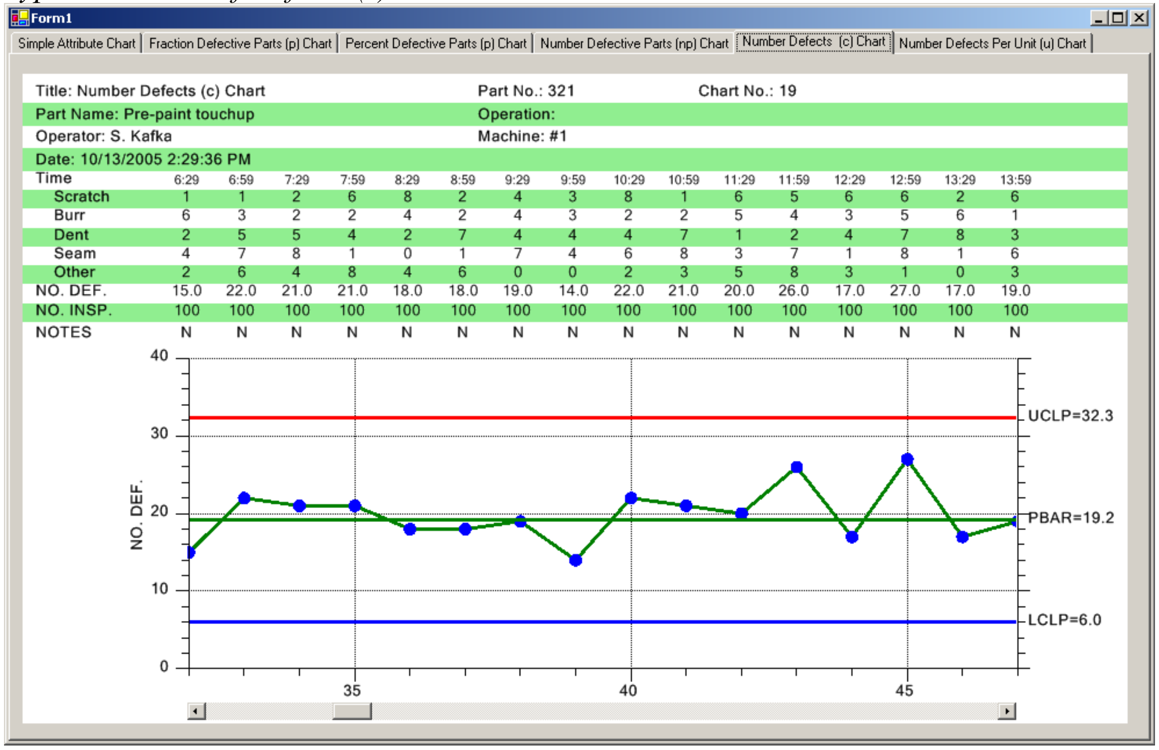
Defect and Defect Category Data Tables

As discussed under the *Variable Control Chart*section, standard worksheets used to gather and plot SPC data consist of three main parts.

- The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.
- The second part records the defect data, organized as a table recording the defect data and SPC calculations in a neat, readable fashion.
- The third part plots the calculated SPC values in the actual SPC chart.

The *Attribute Control Chart* templates that we have created have options that enable the programmer to customize and automatically include header information along with a table of the defect data, organized by defect category, number of defective parts, or total number of defects. Enable the scrollbar and you can display the tabular defect data and SPC plots for a window of 8-20 subgroups, from a much larger collection of measurement data representing hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.

Typical Number of Defects (c) Chart with data table


Form1
Simple Attribute Chart
Fraction Defective Parts (p) Chart
Percent Defective Parts (p) Chart
Number Defective Parts (np) Chart
Number Defects (c) Chart
Number Defects Per Unit (u) Chart
40
30
20
10
0
NO. DEF.
35
40
45
UCLP=32.3
PBAR=19.2
LCLP=6.0

Title: Number Defects (c) Chart | Part No.: 321 | Chart No.: 19
Part Name: Pre-paint touchup | Operation:
Operator: S. Kafka | Machine: #1
Date: 10/13/2005 2:29:36 PM

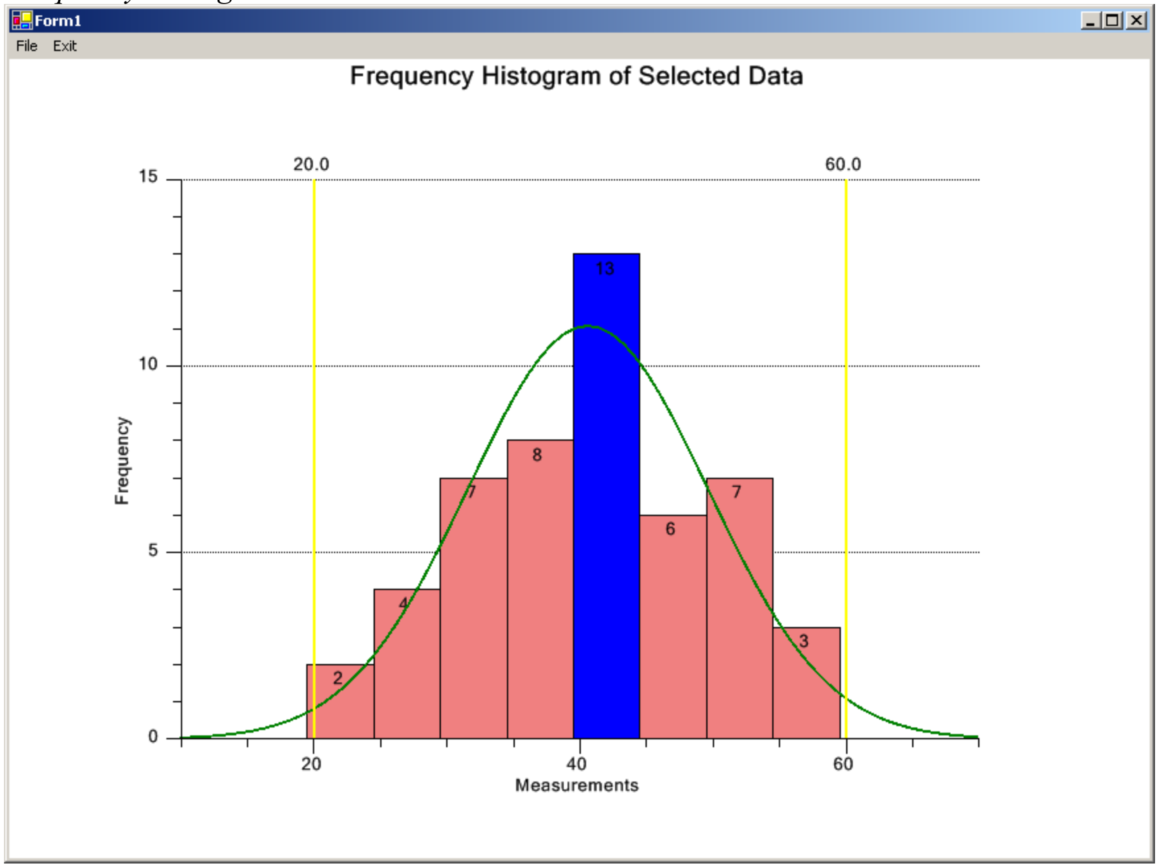
Time	6:29	6:59	7:29	7:59	8:29	8:59	9:29	9:59	10:29	10:59	11:29	11:59	12:29	12:59	13:29	13:59
Scratch	1	1	2	6	8	2	4	3	8	1	6	5	6	6	2	6
Burr	6	3	2	2	4	2	4	3	2	2	5	4	3	5	6	1
Dent	2	5	5	4	2	7	4	4	4	7	1	2	4	7	8	3
Seam	4	7	8	1	0	1	7	4	6	8	3	7	1	8	1	6
Other	2	6	4	8	4	6	0	0	2	3	5	8	3	1	0	3
NO. DEF.	15.0	22.0	21.0	21.0	18.0	18.0	19.0	14.0	22.0	21.0	20.0	26.0	17.0	27.0	17.0	19.0
NO. INSP.	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Other Important SPC Charts

Frequency Histogram Chart

An SPC control chart tacks the trend of critical variables in a production environment. It is important that the production engineer understand whether or not changes or variation in the critical variables are natural variations due to the tolerances inherent to the production machinery, or whether or not the variations are due to some systemic, assignable cause that needs to be addressed. If the changes in critical variables are the result of natural variations, a frequency histogram of the variations will usually follow one of the common continuous (normal, exponential, gamma, Weibull) or discrete (binomial, Poisson, hypergeometric) distributions. It is the job of the SPC engineer to know what distribution best models his process. Periodically plotting of the variation of critical variables will give SPC engineer important information about the current state of the process. A typical frequency histogram looks like:

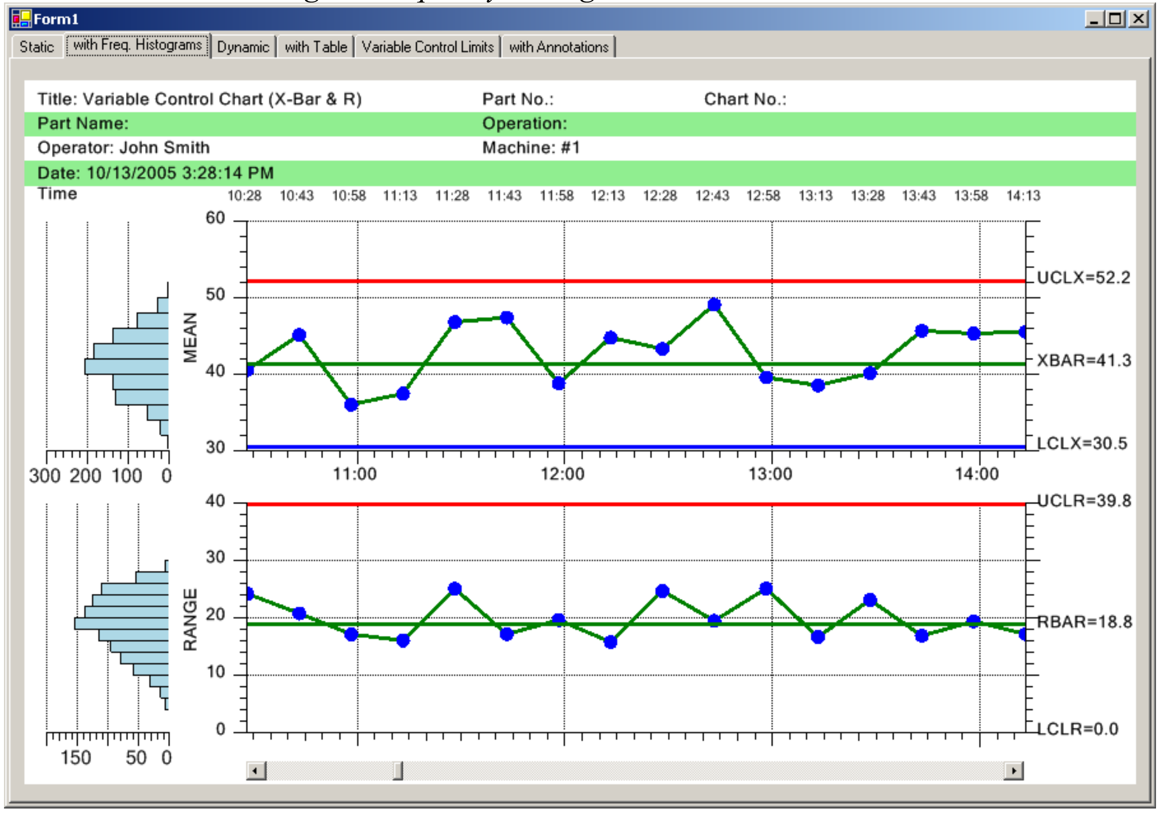
Frequency Histogram Chart



Viewing frequency histograms of both the variation in the primary variable (Mean, Median, count, or actual value), and the secondary variable (Range, Sigma or Moving

Range) side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process.

XBar-R Chart with Integral Frequency Histograms

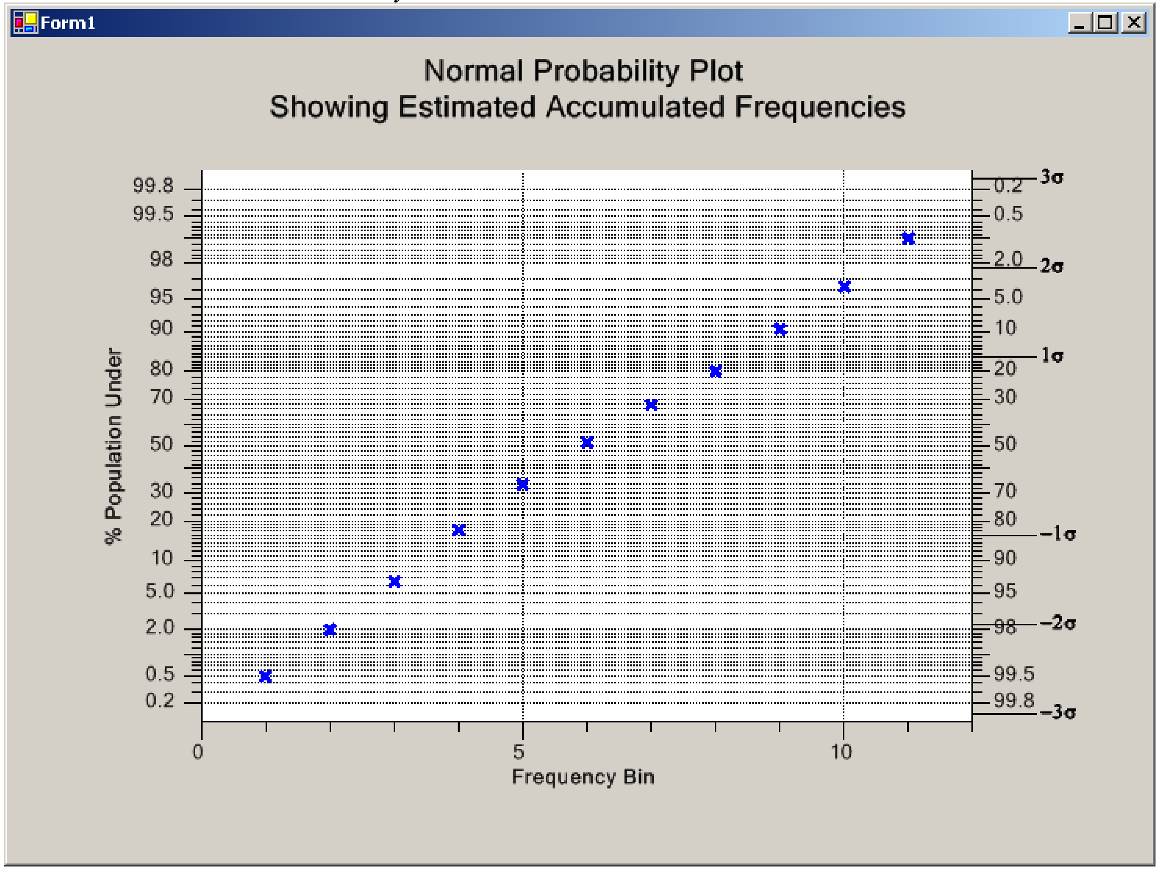


Title: Variable Control Chart (X-Bar & R)
Part No.:
Chart No.:
Part Name:
Operation:
Operator: John Smith
Machine: #1
Date: 10/13/2005 3:28:14 PM

Probability Plots

Another important tool the SPC engineer uses to model the process variation is the probability plot. The probability plot tests whether control chart measurements fit a normal distribution. Usually, the SPC engineer plots probability plot graphs by hand using special probability plot graph paper. We added probability scale and axis classes to the **QCSPCChart** software that plots probability plots directly on the computer. Control chart measurements that follow a normal distribution curve plot as a straight line when plotted in a normal probability plot.

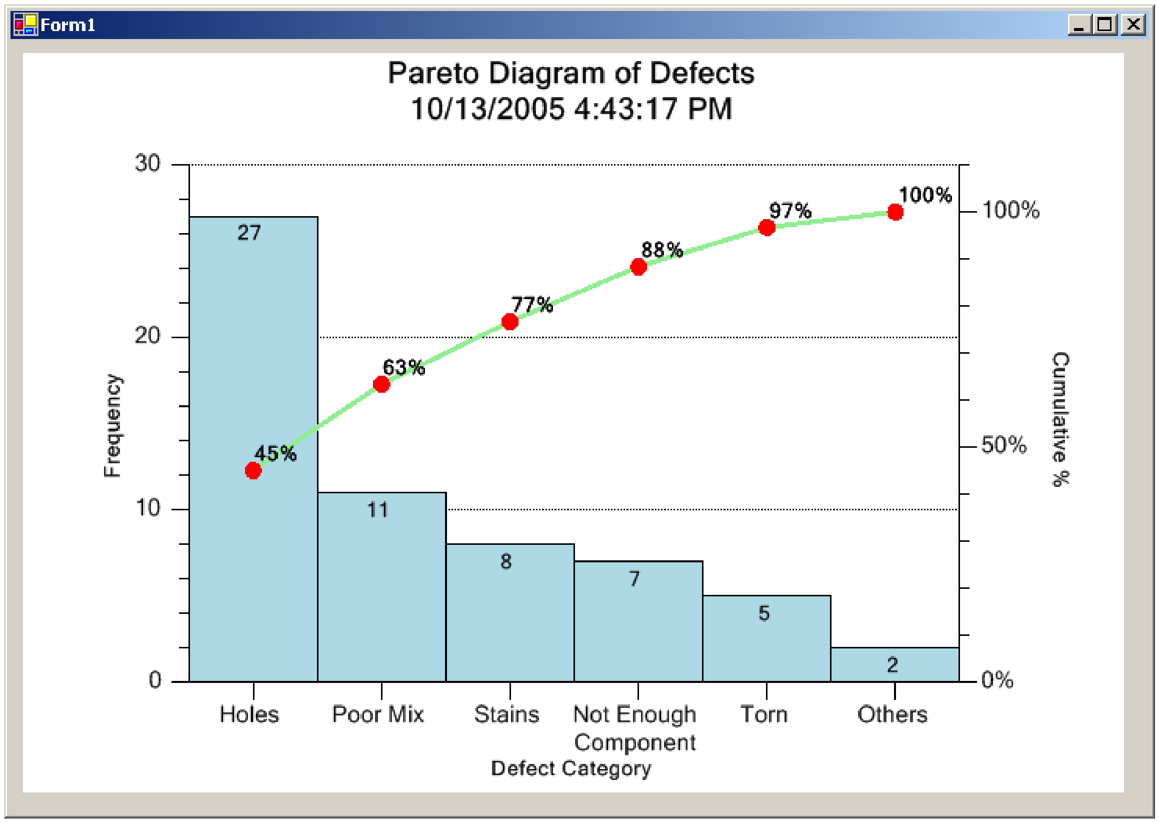
Cumulative Normal Probability Chart



Pareto Diagrams

The Pareto diagram is a special type of bar graph combined with a line plot, plotted in a chart that uses two different y-axis scales. The bar graph scale is a frequency scale that measures the number of times a specific problem has occurred. The line plot scale is a cumulative percentage scale.

Pareto Chart



The chart is easy to interpret. The tallest bar, the left-most one in a Pareto diagram, is the problem that has the most frequent occurrence. The shortest bar, the right-most one, is the problem that has the least frequent occurrence. Time spend on fixing the biggest problem will have the greatest affect on the overall problem rate. This is a simplistic view of actual Pareto analysis, which would usually take into account the cost effectiveness of fixing a specific problem. Never less, it is powerful communication tool that the SPC engineer can use in trying to identify and solve production problems.

3. Class Architecture of the SPC Control Chart Tools for *Java* Class Library

Major Design Considerations

This chapter presents an overview of the **SPC Control Chart Tools for *Java*** class architecture. It discusses the major design considerations of the architecture:

Major design consideration specific to **SPC Control Chart Tools for *Java*** are:

⑤ Direct support for the following SPC chart types:

Variable Control Charts

Fixed sample size subgroup control charts

- X-Bar R – (Mean and Range) chart
- X-Bar Sigma (Mean and Sigma) chart
- Median and Range (Median and Range) chart
- X-R (Individual Range Chart) chart
- EWMA (Exponentially Weighted Moving Average Chart)
- MA (Moving Average Chart)
- MAMR (Moving Average/Moving Range)
- MAMS (Moving Average/Moving Sigma)
- CuSum (Tabular Cumulative Sum Chart)

Variable sample size subgroup control charts

- X-Bar Sigma (Mean and Sigma Chart)

Attribute Control Charts

Fixed sample size subgroup control charts

- p-Chart (Fraction or Percent of Defective Parts, Fraction or Percent Non-Conforming)
- np Chart (Number of Defective Parts, Number of Non-Conforming)
- c-Chart (Number of Defects, Number of Non-Conformities)
- u-Chart (Number of Defects per Unit, Number of Non-Conformities Per Unit)
- DPMO (Number of Defects per Million)

Variable sample size subgroup control charts

- p Chart (Fraction or Percent of Defective Parts)
- u-Chart (Number of Defects per Unit)

SPC Analysis Charts

Frequency Histograms
Probability Charts
Pareto Charts

- Minimal programming required – create SPC charts with a few lines of code using our SPC chart templates.

- Integrated frequency histograms support – Display frequency histograms of sampled data, displayed side-by-side, sharing the same y-axis, with the SPC chart.

- Charts Header Information – Customize the chart display with job specific information, for example: Title, Operator, Part Number, Specification Limits, Machine, ect.

- Table display of SPC data – Display the sampled and calculated values for a SPC chart in a table, directly above the associated point in the SPC chart, similar to standardized SPC worksheets.

- Automatic calculation of SPC control limits – Automatically calculate SPC control limits using sampled data, using industry standard SPC control limit algorithms unique to each chart type.

- Automatic y-Axis scaling – Automatically calculated the y-axis scale for SPC charts, taking into account sampled and calculated data points, and any control limit lines added to the graph.

- Alarms – When monitored value exceeds a SPC control limit it can trigger an event that vectors to a user-written alarm processing delegate.

- Notes – The operator can view or enter notes specific to a specific sample subgroup using a special notes tooltip.

- Data tooltips – The operator can view chart data values using a simple drill-down data tooltip display. The Data tooltips can optionally display sample subgroup data values and statistics, including process capability calculations (Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk) and any notes that have been entered for the sample subgroup.

- Data dogging – SPC data (time stamp and/or batch number, sample values, calculated values, control limit values, and notes can be logged to disk in a CSV (commas separated value) file format.

- Scrollable view – Enable the scroll bar option and scroll through the chart and table view of the SPC data for an unlimited number of sample subgroups.

⑤ Other, optional features – There are many optional features that SPC charts often use, including:

> Multiple SPC control limits, corresponding to +-1, 2 and 3 sigma limits.
>
> Scatter plots of all sampled data values on top of calculated means and medians.
>
> Data point annotations

The chapter also summarizes the classes in the **SPC Control Chart Tools for *Java*** library.

SPC Control Chart Tools for *Java* Class Summary

The **SPC Control Chart Tools for *Java*** library is a super set of the **QCChart2D** library. The classes of the **QCChart2D** library are an integral part of the software. A summary of the **QCChart2D** classes appears below.

QCChart2D Class Summary

Chart view class	The chart view class is a JPanel subclass that manages the graph objects placed in the graph
Data classes	There are data classes for simple xy and group data types. There are also data classes that handle System.DateTime date/time data and contour data.
Scale transform classes	The scale transform classes handle the conversion of physical coordinate values to working coordinate values for a single dimension.
Coordinate transform classes	The coordinate transform classes handle the conversion of physical coordinate values to working coordinate values for a parametric (2D) coordinate system.
Attribute class	The attribute class encapsulates the most common attributes (line color, fill color, line style, line thickness, etc.) for a chart object.
Auto-Scale classes	The coordinate transform classes use the auto-scale classes to establish the minimum and maximum values used to scale a 2D coordinate system. The axis classes also use the auto-scale classes to establish proper tick mark spacing values.

Charting object classes	The chart object classes includes all objects placeable in a chart. That includes axes, axes labels, plot objects (line plots, bar graphs, scatter plots, etc.), grids, titles, backgrounds, images and arbitrary shapes.
Mouse interaction classes	These classes, directly and indirectly System.EventHandler delegates that trap mouse events and permit the user to create and move data cursors, move plot objects, display tooltips and select data points in all types of graphs.
File and printer rendering	These classes render the chart image to a printer, to a variety of file formats including JPEG, and BMP, or to a Java Image object.
Miscellaneous utility classes	Other classes use these for data storage, file I/O, and data processing.

For each of these categories see the associated description in the **QCChart2D** manual (QCChart2DJavaManual.pdf). The **SPC Control Chart Tools for *Java*** classes are in addition to the ones above. They are summarized below.

SPC Control Chart Tools for Java Class Hierarchy

The **QCSPCChart** classes are a super set of the **QCChart2D** charting software. No attempt should be made to utilize the **QCSPCChart** classes without a good understanding of the **QCChart2D** classes. See the **QCChart2DJavaManual** PDF file for detailed information about the **QCChart2D** classes. The diagram below depicts the class hierarchy of the **SPC Control Chart Tools for *Java*** library without the additional **QCChart2D** classes

Namespace com.quinncurtis.spcchartjava.

- com.quinn-curtis.chart2djava.ChartView
  - FrequencyHistogramChart
  - ParetoChart
  - ProbabilityChart
  - SPCChartBase
    - SPCBatchAttributeControlChart
    - SPCBatchVariableControlChart
    - SPCTimeAttributeControlChart
    - SPCTimeVariableControlChart

- com.quinncurtis.chart2djava.AutoScale
  - ProbabilityAutoScale
- com.quinncurtis.chart2djava.Axis

    ProbabilityAxis
com.quinncurtis.chart2djava.LinearAxis
    ProbabilitySigmaAxis
com.quinncurtis.chart2djava.PhysicalCoordinates
    ProbabilityCoordinates
com.quinncurtis.chart2djava.Scale
    ProbabilityScale
com.quinncurtis.chart2djava.StringLabel
    NotesLabel
com.quinncurtis.chart2djava.MouseListener
    NotesToolTip
com.quinncurtis.chart2djava.DataToolTip
    SPCDataToolTip

QCSPCChart Classes

SPCControlChartData
SPCControlLimitAlarmArgs
SPCControlLimitRecord
SPCCalculatedValueRecrod
SPCProcessCapabilityRecord
SPCSampledValueRecord
SPCControlParameters
SPCGeneralizedTableDisplay
SPCControlPlotObjects
SPCChartObjects

SPC Control Chart Data

SPCControlChartData

SPC control chart data is stored in the **SPCControlChartData** class. It holds the header information used to customize the chart table, the raw sample data used to prepare the chart, the calculated chart values used in the chart, and the SPC control limits. It contains array lists of **SPCSampledValueRecord**, **SPCControlLimitRecord** and **SPCCalculatedValueRecord** objects.

SPCSampledValueRecord

This class encapsulates a sample data value. It includes a description for the item, the current value of the sampled value, and a history of previous values.

SPCControlLimitRecord

This class holds information specific to a SPC control limit: including the current value of the control limit, a history of control limit values, description, and the hysteresis value for alarm checking.

SPCCalculatedValueRecord

The record class for a calculated SPC statistic. It holds the calculated value type (mean, median, sum, variance, standard deviation, etc.), value, description and historical data.

SPCProcessCapabilityRecord

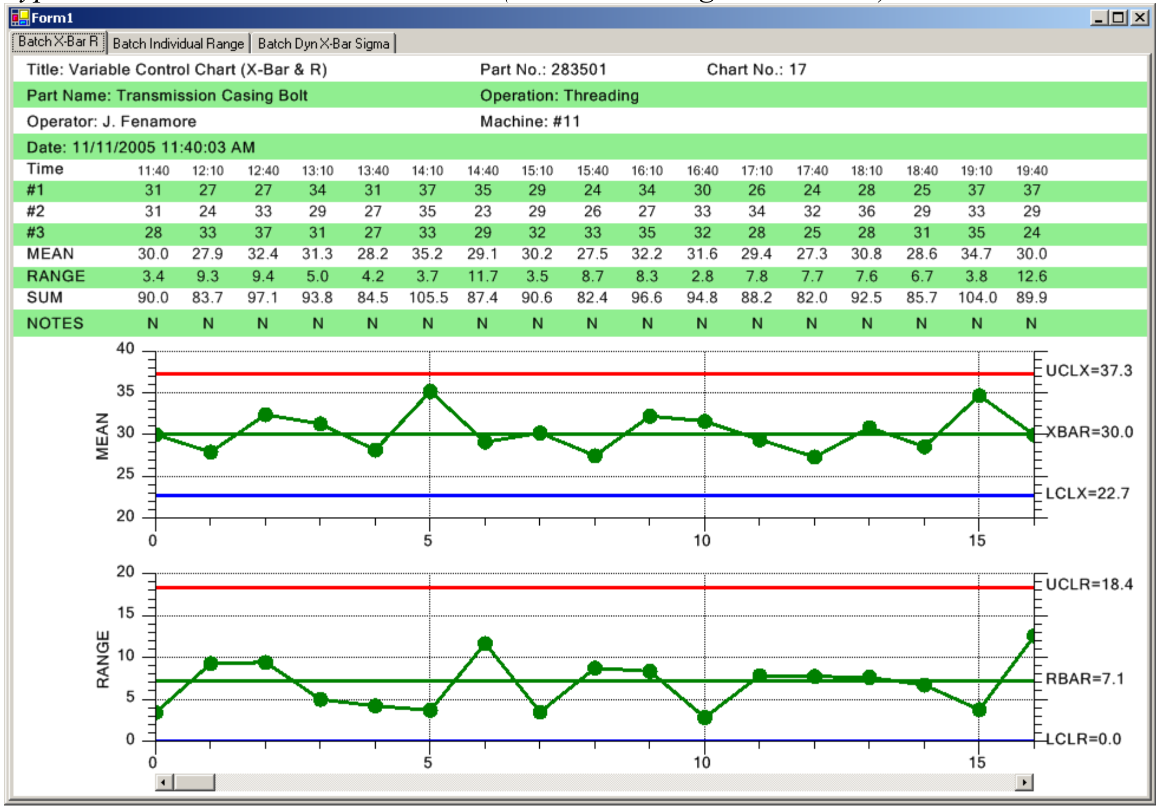
The record class for storing and calculating process capability statistics: Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu and Ppk.

SPC Charts and Related Chart Objects

SPCChartBase

The **SPCChartBase** forms the base object for all SPC control charts. The variable control chart templates (**SPCBatchVariableControlChart** and **SPCTimeVariableControlChart**), and attribute control charts (**SPCBatchAttributeControlChart** and **SPCTimeAttributeControlChart**) are derived from the **SPCChartBase** class.

Typical Batch Variable Control Chart (Mean and Range or X-Bar R)



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Fenamore Machine: #11

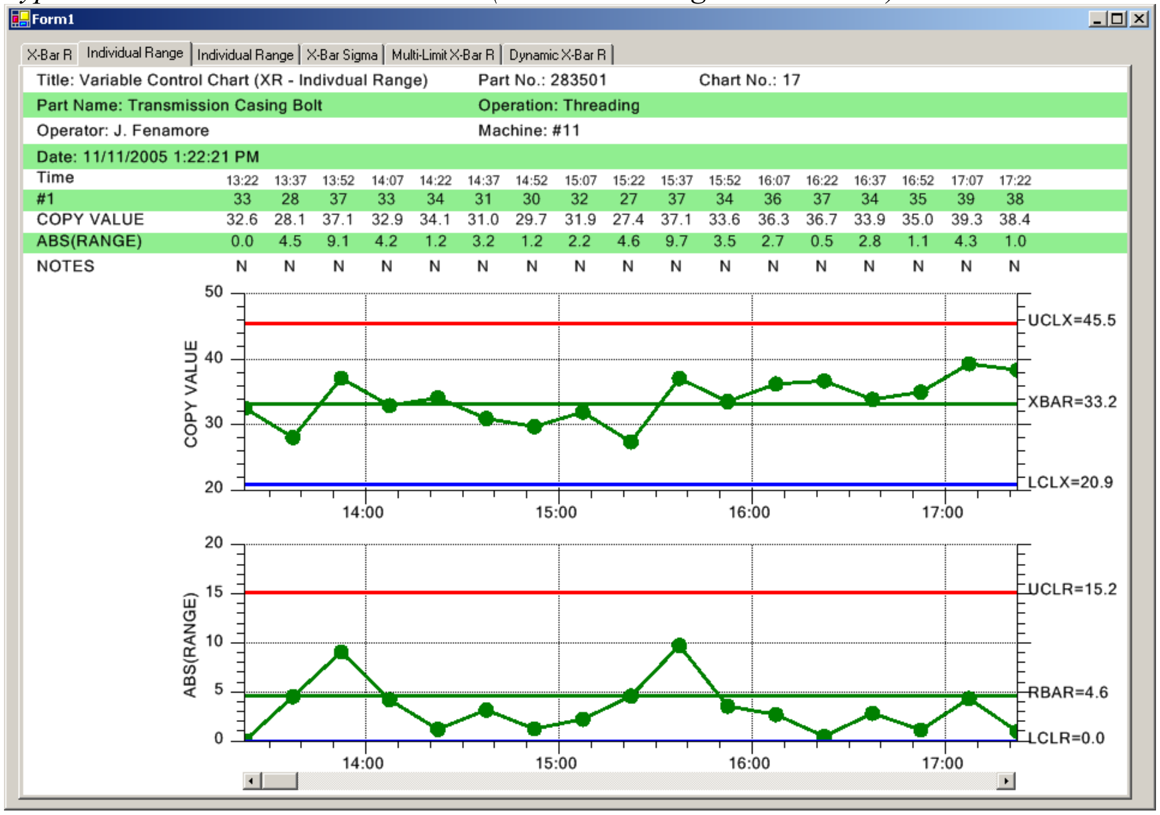
Date: 11/11/2005 11:40:03 AM

Time	11:40	12:10	12:40	13:10	13:40	14:10	14:40	15:10	15:40	16:10	16:40	17:10	17:40	18:10	18:40	19:10	19:40
#1	31	27	27	34	31	37	35	29	24	34	30	26	24	28	25	37	37
#2	31	24	33	29	27	35	23	29	26	27	33	34	32	36	29	33	29
#3	28	33	37	31	27	33	29	32	33	35	32	28	25	28	31	35	24
MEAN	30.0	27.9	32.4	31.3	28.2	35.2	29.1	30.2	27.5	32.2	31.6	29.4	27.3	30.8	28.6	34.7	30.0
RANGE	3.4	9.3	9.4	5.0	4.2	3.7	11.7	3.5	8.7	8.3	2.8	7.8	7.7	7.6	6.7	3.8	12.6
SUM	90.0	83.7	97.1	93.8	84.5	105.5	87.4	90.6	82.4	96.6	94.8	88.2	82.0	92.5	85.7	104.0	89.9
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

SPCBatchVariableControlChart

A *Batch Variable Control Chart* class that uses a **CartesianCoordinate** system with a numeric based X-Axis. This class creates MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART and MAMS_CHART chart types.

Typical Time Variable Control Chart (Individual Range or XR Chart)



Title: Variable Control Chart (XR - Indivdual Range)   Part No.: 283501   Chart No.: 17

Part Name: Transmission Casing Bolt   Operation: Threading

Operator: J. Fenamore   Machine: #11

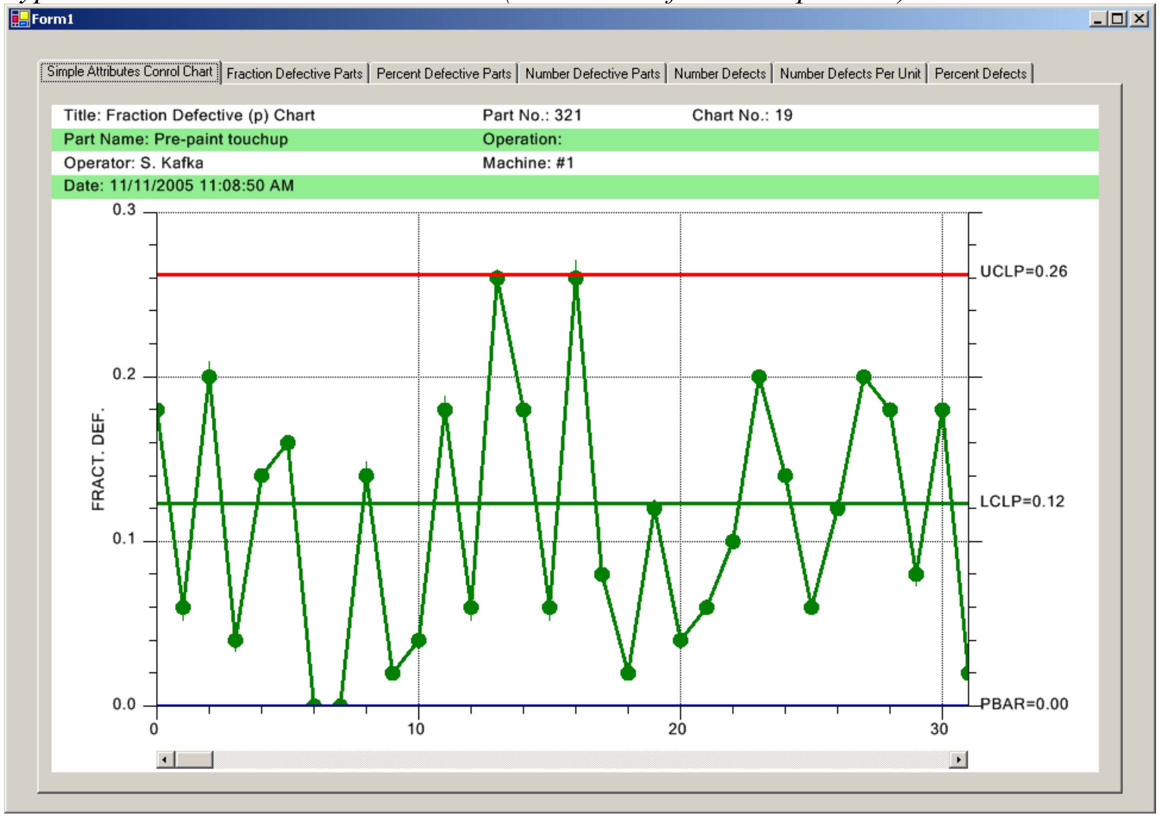
Date: 11/11/2005 1:22:21 PM

Time	13:22	13:37	13:52	14:07	14:22	14:37	14:52	15:07	15:22	15:37	15:52	16:07	16:22	16:37	16:52	17:07	17:22
#1	33	28	37	33	34	31	30	32	27	37	34	36	37	34	35	39	38
COPY VALUE	32.6	28.1	37.1	32.9	34.1	31.0	29.7	31.9	27.4	37.1	33.6	36.3	36.7	33.9	35.0	39.3	38.4
ABS(RANGE)	0.0	4.5	9.1	4.2	1.2	3.2	1.2	2.2	4.6	9.7	3.5	2.7	0.5	2.8	1.1	4.3	1.0
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

SPCTimeVariableControlChart

A *Variable Control Chart* class that uses a **TimeCoordinate** system with a time based X-Axis. This class creates MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART and MAMS_CHART chart types.

Typical Batch Attribute Control Chart (Fraction Defective or p-Chart)

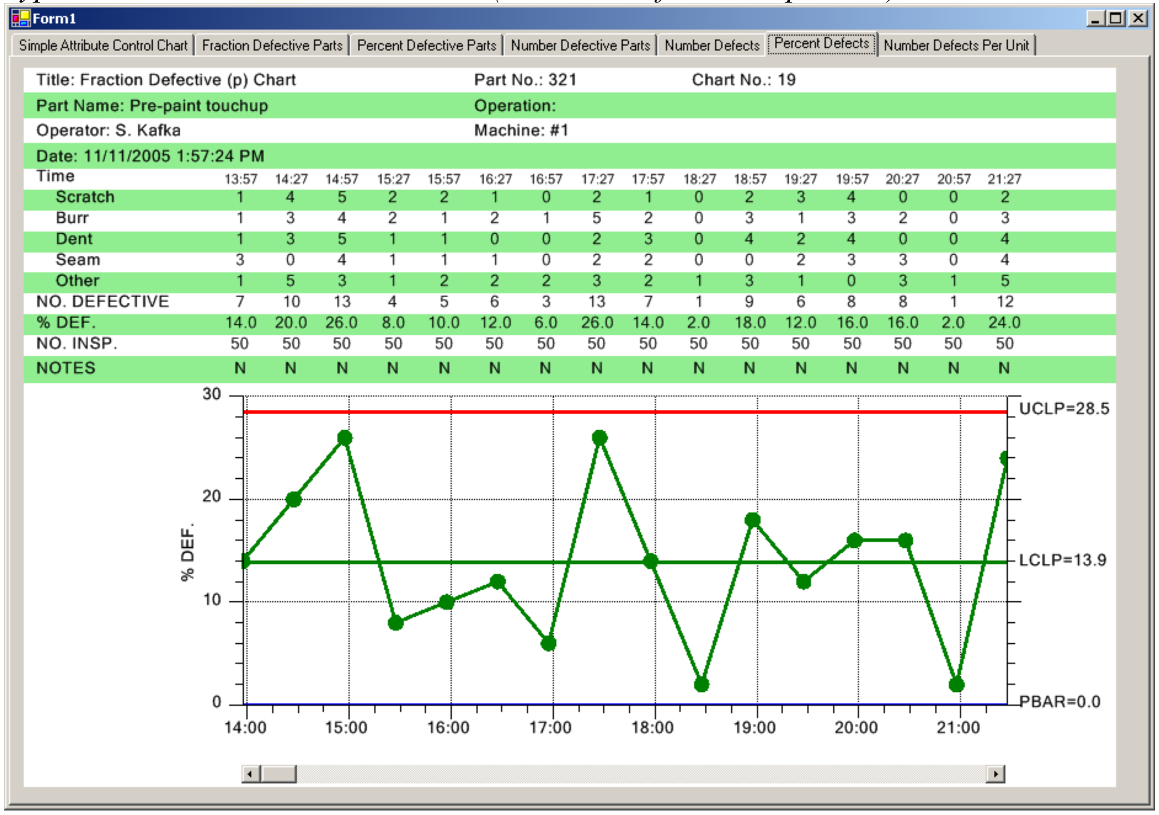


Title: Fraction Defective (p) Chart Part No.: 321 Chart No.: 19
Part Name: Pre-paint touchup Operation:
Operator: S. Kafka Machine: #1
Date: 11/11/2005 11:08:50 AM

SPCBatchAttributeControlChart

A Batch Attribute Control Chart class that uses a **CartesianCoordinate** system with a numeric X-Axis. This class creates PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART SPC, NUMBER_DEFECTS_PER_MILLION_CHART, PERCENT_DEFECTIVE_PARTS_CHART_VSS, FRACTION_DEFECTIVE_PARTS_CHART_VSS, NUMBER_DEFECTS_PERUNIT_CHART_VSS chart types.

Typical Time Attribute Control Chart (Fraction Defective or p-Chart)



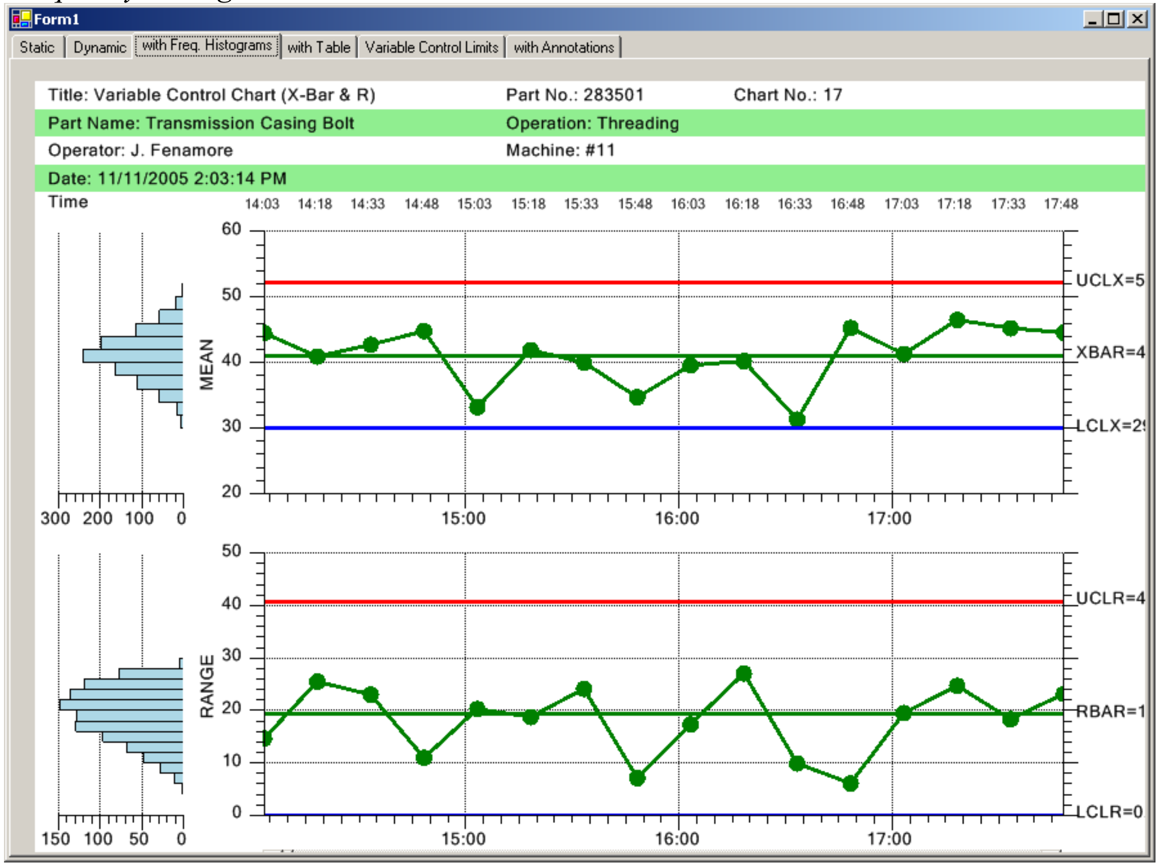
Title: Fraction Defective (p) Chart Part No.: 321 Chart No.: 19
Part Name: Pre-paint touchup Operation:
Operator: S. Kafka Machine: #1
Date: 11/11/2005 1:57:24 PM

Time	13:57	14:27	14:57	15:27	15:57	16:27	16:57	17:27	17:57	18:27	18:57	19:27	19:57	20:27	20:57	21:27
Scratch	1	4	5	2	2	1	0	2	1	0	2	3	4	0	0	2
Burr	1	3	4	2	1	2	1	5	2	0	3	1	3	2	0	3
Dent	1	3	5	1	1	0	0	2	3	0	4	2	4	0	0	4
Seam	3	0	4	1	1	1	0	2	2	0	0	2	3	3	0	4
Other	1	5	3	1	2	2	2	3	2	1	3	1	0	3	1	5
NO. DEFECTIVE	7	10	13	4	5	6	3	13	7	1	9	6	8	8	1	12
% DEF.	14.0	20.0	26.0	8.0	10.0	12.0	6.0	26.0	14.0	2.0	18.0	12.0	16.0	16.0	2.0	24.0
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

SPCTimeAttributeControlChart

An *Attribute Control Chart*class that uses a **TimeCoordinate** system with a time based X-Axis This class creates PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART, NUMBER_DEFECTS_PER_MILLION_CHART, PERCENT_DEFECTIVE_PARTS_CHART_VSS, FRACTION_DEFECTIVE_PARTS_CHART_VSS, NUMBER_DEFECTS_PERUNIT_CHART_VSS chart types.

Frequency Histograms used in Combination with a SPC Control Chart

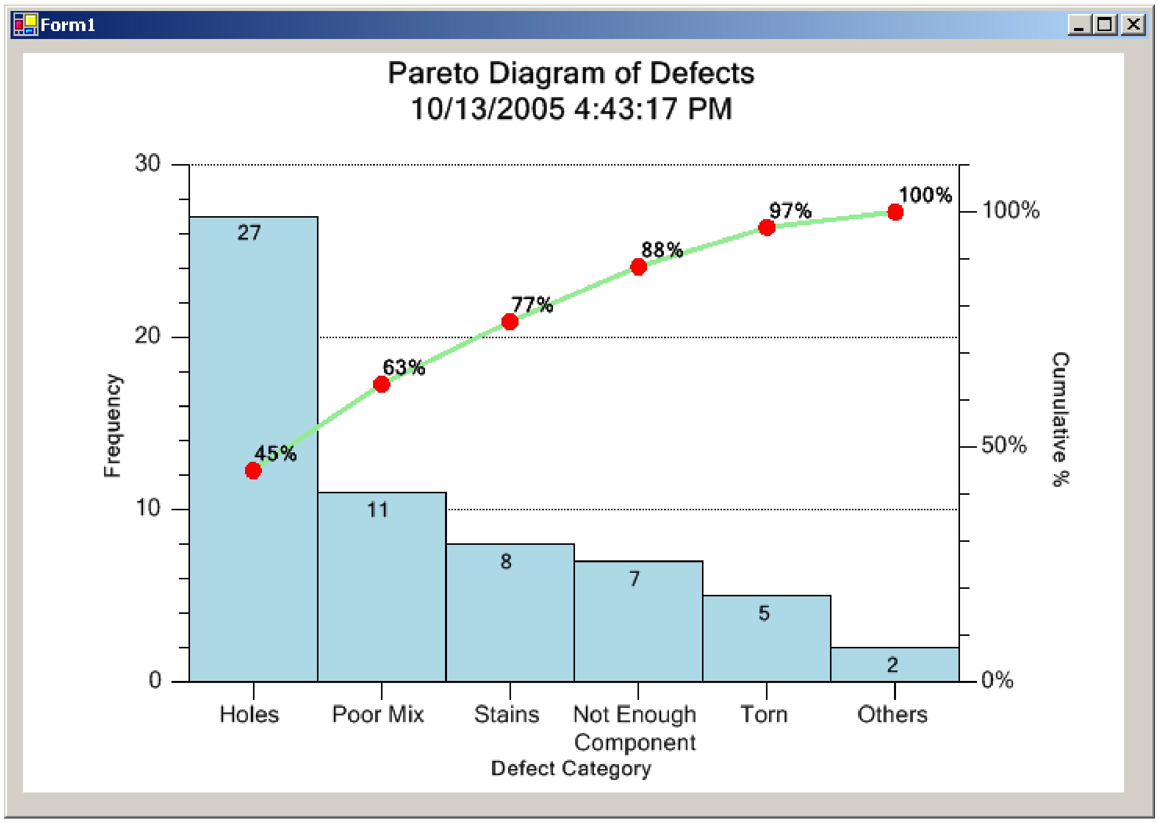


Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 11/11/2005 2:03:14 PM
Time 14:03 14:18 14:33 14:48 15:03 15:18 15:33 15:48 16:03 16:18 16:33 16:48 17:03 17:18 17:33 17:48

FrequencyHistogramChart

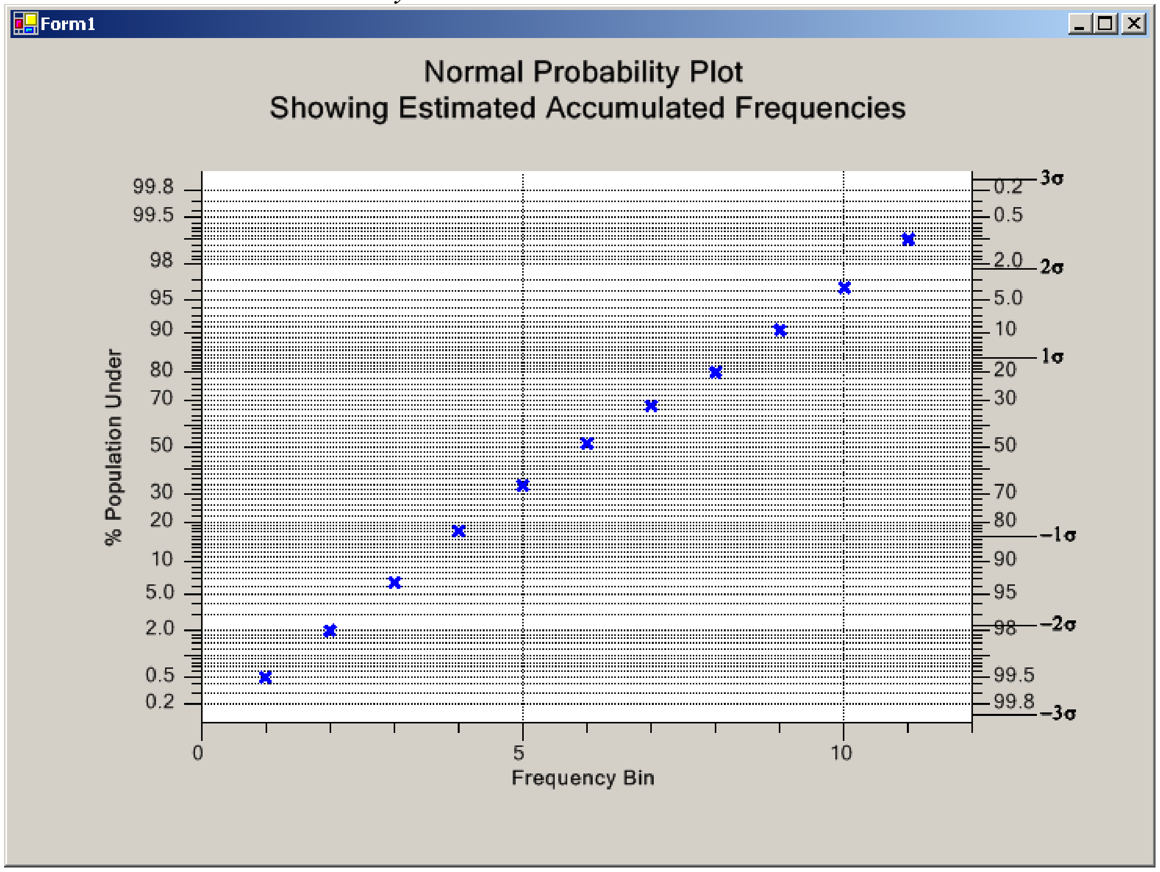
A *Frequency Histogram* checks that the variation in a process variable follows the predicted distribution function (normal, Poisson, chi-squared, etc). The class includes all of the objects needed to draw a complete frequency histogram chart. These objects include objects for data, a coordinate system, titles, axes, axes labels, grids and a bar plot.

Pareto Chart



ParetoChart The *Pareto Diagram* is special type of bar graph combined with a line plot, plotted in a chart that uses two different y-axis scales. The bar graph scale is a frequency scale that measures the number of times a specific problem has occurred. The line plot scale is a cumulative percentage scale. The class includes all of the objects needed to draw a complete Pareto chart. These objects include objects for data, coordinate systems, titles, axes, axes labels, grids, numeric labels, and a line plot and bar plot.

Cumulative Normal Probability Chart



Probability Plots The **ProbabilityChart** class is a highly specialized chart template used to plot cumulative frequency data using a coordinate system that has a cumulative probability y-scale. The class includes all of the objects needed to draw a complete Probability chart. These objects include objects for data, coordinate systems, titles, axes, axes labels, grids, numeric labels, and scatter plot. New classes were developed for the **QCChart2D** charting software capable of rendering of probability chart coordinate systems (**ProbabilityScale**, **ProbabilityAutoScale**, **ProbabilityCoordinates**) and probability axes (**ProbabilityAxis**, **ProbabilitySigmaAxis**).

ProbabilityScale The **ProbabilityScale** class implements a cumulative normal probability coordinate system for a single coordinate, x or y. Two such scales provide the scaling routines for x and y in an **PhysicalCoordindates** derived class, **CartesianCoordinates**, for example. This allows for different x and y scale types (linear, cumulative normal probability, time) to be installed independently for x- and y-coordinates.

ProbabilityAutoScale

The **ProbabilityAutoScale** class is used with cumulative normal probability coordinates and auto-scales the plotting area of graphs and to set the minimum and maximum values of the axes displayed in the graphs.

ProbabilityCoordinates

The **ProbabilityCoordinates** class extends the **PhysicalCoordinates** class to support a y-axis probability scale in an xy coordinate plane.

ProbabilityAxis The **ProbabilityAxis** class implements a probability axis where the major tick marks are placed at intervals appropriate to a cumulative probability scale.

ProbabilitySigmaAxis

The **ProbabilitySigmaAxis** class implements a linear axis where the tick marks are placed at linear intervals on the sigma levels of the associated probability scale.

NotesLabel The **NotesLabel** class displays the Notes items in the SPC table.

NotesToolTip The **NotesToolTip** displays the Notes tooltip for the notes items in the SPC table.

SPCDataToolTip The **SPCDataTooTip** displays the data tooltip for SPC Charts

SPC Calculations

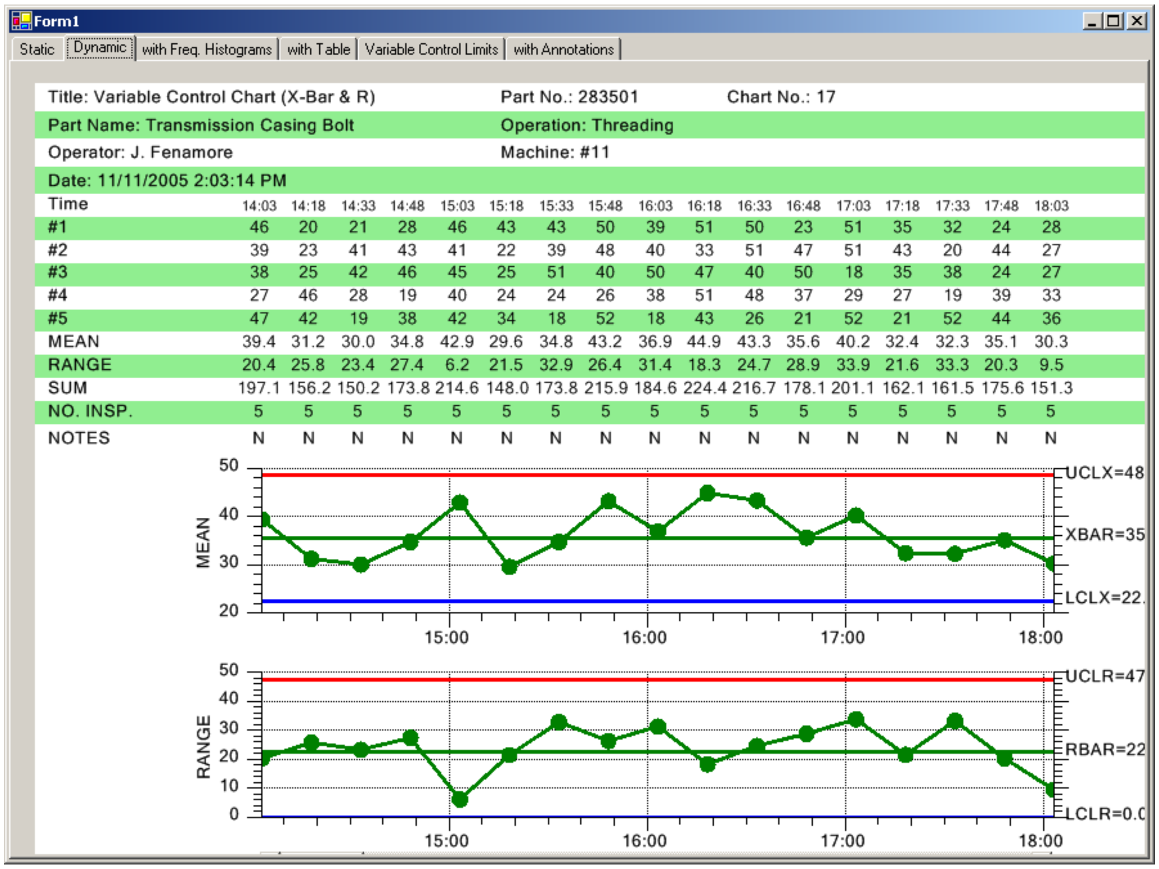
SPCArrayStatistics SPC involves many statistical calculations. The **SPCArrayStatistics** class includes routines for the calculation of sums, means, medians, ranges, minimum values, maximum values, variances, and standard deviations. It also includes routines for array sorting and calculating frequency bins for frequency histograms. It also includes functions that compute cumulative probability values for normal, Poisson, and chi-squared distributions.

SPCControlParameters

The **SPCControlParameters** class contains the factors and formulas for calculating SPC control chart limits for *Variable* and *Attribute Control Charts*. It includes calculations for the most common SPC charts: X-Bar R, Median and Range, X-Bar Sigma, X-R, u-chart, p-chart, np-chart, and c-chart.

Tabular Display

Table Display of Sampled and Calculated SPC Control Chart Value



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 11/11/2005 2:03:14 PM

Time	14:03	14:18	14:33	14:48	15:03	15:18	15:33	15:48	16:03	16:18	16:33	16:48	17:03	17:18	17:33	17:48	18:03
#1	46	20	21	28	46	43	43	50	39	51	50	23	51	35	32	24	28
#2	39	23	41	43	41	22	39	48	40	33	51	47	51	43	20	44	27
#3	38	25	42	46	45	25	51	40	50	47	40	50	18	35	38	24	27
#4	27	46	28	19	40	24	24	26	38	51	48	37	29	27	19	39	33
#5	47	42	19	38	42	34	18	52	18	43	26	21	52	21	52	44	36
MEAN	39.4	31.2	30.0	34.8	42.9	29.6	34.8	43.2	36.9	44.9	43.3	35.6	40.2	32.4	32.3	35.1	30.3
RANGE	20.4	25.8	23.4	27.4	6.2	21.5	32.9	26.4	31.4	18.3	24.7	28.9	33.9	21.6	33.3	20.3	9.5
SUM	197.1	156.2	150.2	173.8	214.6	148.0	173.8	215.9	184.6	224.4	216.7	178.1	201.1	162.1	161.5	175.6	151.3
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

SPCGeneralizedTableDisplay

The **SPCGeneralizedTableDisplay** manages a list of **ChartText** objects (**NumericLabel**, **StringLabel** and **TimeLabel** objects), that encapsulate each unique table entry in the SPC chart table. This class also manages the spacing between the rows and columns of the table, and the alternating stripe used as a background for the table.

SPC Control Alarms

SPCControlLimitAlarmArgs

This class passes event information to a **SPCControlLimitAlarmEventDelegate** alarm processing delegate.

SPCControlLimitAlarmEventDelegate

A delegate type for hooking up control limit alarm notifications

4. QCChart2D for *Java* Class Summary

Chart view class	The chart view class is a **JPanel** subclass that manages the graph objects placed in the graph
Data classes	There are data classes for simple xy and group data types. There are also data classes that handle **GregorianCalendar** date/time data and contour data.
Scale transform classes	The scale transform classes handle the conversion of physical coordinate values to working coordinate values for a single dimension.
Coordinate transform classes	The coordinate transform classes handle the conversion of physical coordinate values to working coordinate values for a parametric (2D) coordinate system.
Attribute class	The attribute class encapsulates the most common attributes (line color, fill color, line style, line thickness, etc.) for a chart object.
Auto-Scale classes	The coordinate transform classes use the auto-scale classes to establish the minimum and maximum values used to scale a 2D coordinate system. The axis classes also use the auto-scale classes to establish proper tick mark spacing values.
Charting object classes	The chart object classes includes all objects placeable in a chart. That includes axes, axes labels, plot objects (line plots, bar graphs, scatter plots, etc.), grids, titles, backgrounds, images and arbitrary shapes.
Mouse interaction classes	These classes, directly and indirectly derived from the Java **MouseInputListener** class, trap mouse events and permit the user to create and move data cursors, move plot objects, display tooltips and select data points in all types of graphs.
File and printer rendering	These classes render the chart image to a printer, to a JPEG file, or to a Java **Image** object.
Miscellaneous utility classes	Other classes use these for data storage, file I/O, and data processing.

A summary of each category appears in the following section.

Chart Window Classes

javax.swing.JPanel
- **ChartView**

The starting point of a chart is the **ChartView** class. The **ChartView** class derives from the Java Swing **JPanel** class**,** where the **JPanel** class is the base class for the Swing collection of standard components such as menus, buttons, check boxes, etc. The **ChartView** class manages a collection of chart objects in a chart and automatically updates the chart objects when the underlying window processes a paint event. Since the **ChartView** class is a subclass of the **JPanel** class, it acts as a container for other Java components too. Use a layout manager (including the null layout manager) to position Java components in the **ChartView** window.

Data Classes

ChartDataset
- **SimpleDataset**
  - **TimeSimpleDataset**
  - **ElapsedTimeSimpleDataset**
  - **ContourDataset**
  - **EventSimpleDataset**
- **GroupDataset**
  - **TimeGroupDataset**
  - **ElapsedTimeGroupDataset**
  - **EventGroupDataset**

The dataset classes organize the numeric data associated with a plotting object. There are two major types of data supported by the **ChartDataset** class. The first is simple xy data, where for every x-value there is one y-value. The second data type is group data, where every x-value can have one or more y-values.

ChartDataset	The abstract base class for the other dataset classes. It contains data common to all of the dataset classes, such as the x-value array, the number of x-values, the dataset name and the dataset type.

SimpleDataset	Represents simple xy data, where for every x-value there is one y-value.
TimeSimpleDataset	A subclass of **SimpleDataset**, it is initialized using **ChartCalendar** dates (a wrapper around the System.DateTime value class) in place of the x- or y-values.
ElapsedTimeSimpleDataset	A subclass of **SimpleDataset**, it is initialized with **ChartTimeSpan** objects, or milliseconds, in place of the x- or y-values.
EventSimpleDataset	A subclass of **SimpleDataset**, it is initialized with **ChartEvent** objects, where the data is to be plotted using one of the simple plot types.
ContourDataset	A subclass of **SimpleDataset**, it adds a third dimension (z-values) to the x- and y- values of the simple dataset.
GroupDataset	Represents group data, where every x-value can have one or more y-values.
TimeGroupDataset	A subclass of **GroupDataset**, it uses **ChartCalendar** dates (a wrapper around the System.DateTime value class) as the x-values, and floating point numbers as the y-values.
ElapsedTimeGroupDataset	A subclass of **GroupDataset**, it uses **ChartTimeSpan** objects, or milliseconds, as the x-values, and floating point numbers as the y-values.
EventGroupDataset	A subclass of **GroupDataset**, it uses **ChartEvent** objects, where the data is to be plotted using one of the group plot types.

Scale Classes

ChartScale
- **LinearScale**
- **LogScale**
- **TimeScale**
- **ElapsedTimeScale**
- **EventScale**

The **ChartScale** abstract base class defines coordinate transformation functions for a single dimension. It is useful to be able to mix and match different scale transform functions for x- and y-dimensions of the **PhysicalCoordinates** class. The job of a **ChartScale** derived object is to convert a dimension from the current *physical* coordinate system into the current *working* coordinate system.

LinearScale	A concrete implementation of the **ChartScale** class. It converts a linear physical coordinate system into the working coordinate system.
LogScale	A concrete implementation of the **ChartScale** class. It converts a logarithmic physical coordinate system into the working coordinate system.
TimeScale	A concrete implementation of the **ChartScale** class. converts a date/time physical coordinate system into the working coordinate system.
ElapsedTimeScale	A concrete implementation of the **ChartScale** class. converts an elapsed time coordinate system into the working coordinate system.
EventScale	A concrete implementation of the **ChartScale** class. converts an event coordinate system into the working coordinate system.

Coordinate Transform Classes

- **UserCoordinates**
  - **WorldCoordinates**
    - **WorkingCoordinates**
      - **PhysicalCoordinates**
        - **CartesianCoordinates**
          - **ElapsedTimeCoordinates**
          - **PolarCoordinates**
          - **AntennaCoordinates**
          - **EventCoordinates**
        - **TimeCoordinates**

The coordinate transform classes maintain a 2D coordinate system. Many different coordinate systems are used to position and draw objects in a graph. Examples of some of the coordinate systems include the device coordinates of the current window, normalized coordinates for the current window and plotting area, and scaled physical coordinates of the plotting area.

UserCoordinates	This class manages the interface to the **Graphics2D** classes and contains routines for drawing lines, rectangles and text using Java device coordinates.
WorldCoordinates	This class derives from the **UserCoordinates** class and maps a device independent world coordinate system on top of the Java device coordinate system.
WorkingCoordinates	This class derives from the **WorldCoordinates** class and extends the physical coordinate system of the *plot area* (the area typically bounded by the charts axes) to include the complete *graph area* (the area of the chart outside of the *plot area*).
PhysicalCoordinates	This class is an abstract base class derived from **WorkingCoordinates** and defines the routines needed to map the physical coordinate system of a plot area into a working coordinate system. Different scale objects (**ChartScale** derived) are installed for converting physical x- and y-coordinate values into working coordinate values.
CartesianCoordinates	This class is a concrete implementation of the **PhysicalCoordinates** class and implements a coordinate system used to plot linear, logarithmic and semi-logarithmic graphs.
TimeCoordinates	This class is a concrete implementation of the **PhysicalCoordinates** class and implements a coordinate system used to plot **GregorianCalenar** time-based data.
ElapsedTimeCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot elapsed time data.
PolarCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot polar coordinate data.
AntennaCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot antenna coordinate data. The antenna coordinate system differs from the more common polar coordinate system in that the

	radius can have plus/minus values, the angular values are in degrees, and the angular values increase in the clockwise direction.
EventCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot ChartEvent based data.

Attribute Class

ChartAttribute
ChartGradient

This class consolidates the common line and fill attributes as a single class. Most of the graph objects have a property of this class that controls the color, line thickness and fill attributes of the object. The **ChartGradient** class expands the number of color options available in the **ChartAttribute** class.

ChartAttribute	This class consolidates the common line and fill attributes associated with a **GraphObj** object into a single class.
ChartGradient	A **ChartGradient** can be added to a **ChartAttribute** object, defining a two-color gradient that is applied wherever the color fill attribute is normally used

Auto-Scaling Classes

AutoScale
- **LinearAutoScale**
- **LogAutoScale**
- **TimeAutoScale**
- **ElapsedTimeAutoScale**
- **EventAutoScale**

Usually, programmers do not know in advance the scale for a chart. Normally the program needs to analyze the current data for minimum and maximum values and create a chart scale based on those values. Auto-scaling, and the creation of appropriate axes, with endpoints at even values, and well-rounded major and minor tick mark spacing, is quite complicated. The **AutoScale** classes provide tools that make automatic generation of charts easier.

AutoScale	This class is the abstract base class for the auto-scale classes.
LinearAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **SimpleDataset** and **GroupDataset** objects. Linear scales and axes use it for auto-scale calculations.
LogAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **SimpleDataset** and **GroupDataset** objects. Logarithmic scales and axes use it for auto-scale calculations.
TimeAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the **GregorianCalendar** values in **TimeSimpleDataset** and **TimeGroupDataset** objects. Date/time scales and axes use it for auto-scale calculations.
ElapsedTimeAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **ElapsedTimeSimpleDataset** and **ElapsedTimeGroupDataset** objects. The elapsed time classes use it for auto-scale calculations.
EventAutoScale	This class is a concrete implementation of the **AutoScale**class. It calculates scaling values based on the numeric values in **EventSimpleDataset**and **EventGroupDataset**objects. The evnet classes use it for auto-scale calculations.

Chart Object Classes

Chart objects are graph objects that can be rendered in the current graph window. This is in comparison to other classes that are purely calculation classes, such as the coordinate conversion classes. All chart objects have certain information in common. This includes instances of **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color and line style information for the object, while the

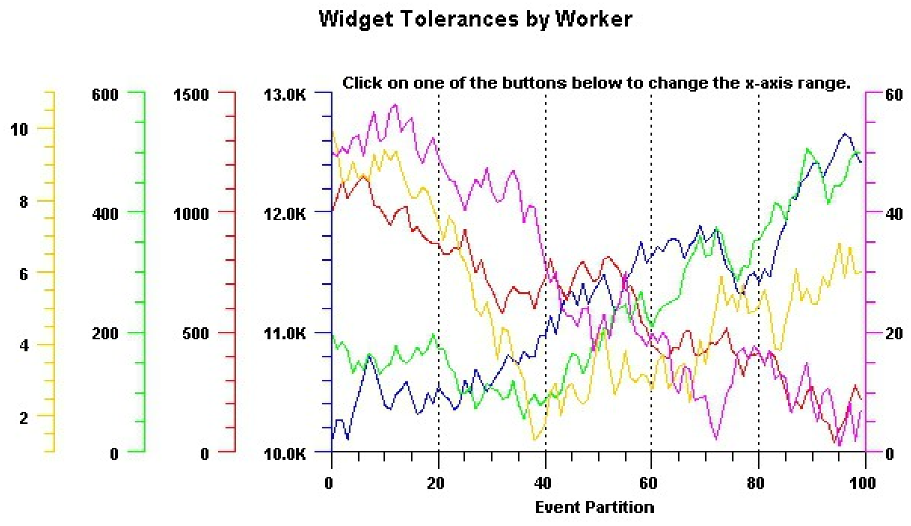
PhysicalCoordinates maintains the coordinate system used by object. The majority of classes in the library derive from the **GraphObj** class, each class a specific charting object such as an axis, an axis label, a simple plot or a group plot. Add **GraphObj** derived objects (axes, plots, labels, title, etc.) to a graph using the **ChartView.addChartObject** method.

GraphObj	This class is the abstract base class for all drawable graph objects. It contains information common to all chart objects. This class includes references to instances of the **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color and line style information for the object, while the **PhysicalCoordinates** maintains the coordinate system used by object. The majority of classes in the library derive from the **GraphObj** class, each class a specific charting object such as an axis, an axis label, a simple plot or a group plot
Background	This class fills the background of the entire chart, or the plot area of the chart, using a solid color, a color gradient, or a texture.

Axis Classes

- **Axis**
  - **LinearAxis**
    - **PolarAxes**
    - **AntennaAxes**
    - **ElapsedTimeAxis**
  - **LogAxis**
  - **TimeAxis**
  - **EventAxis**

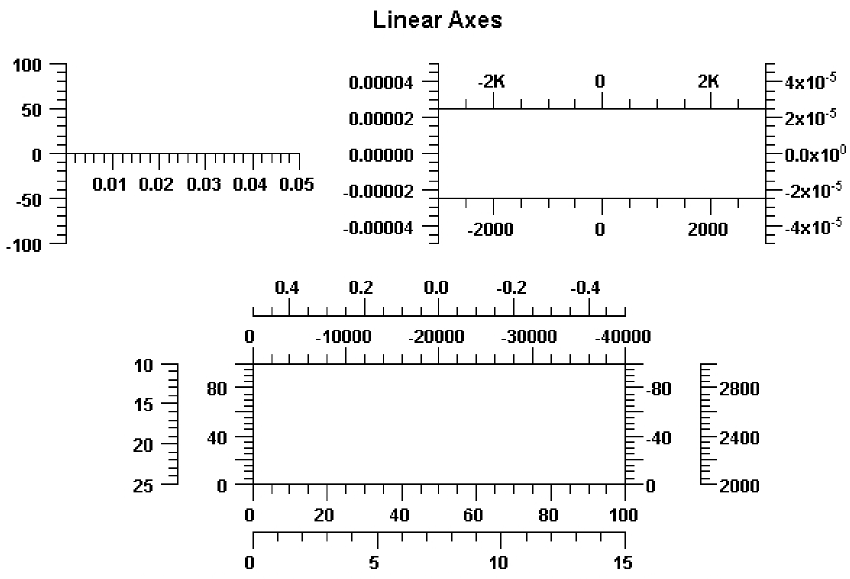
Creating a **PhysicalCoordinates** coordinate system does not automatically create a pair of x- and y-axes. Axes are separate charting objects drawn with respect to a specific **PhysicalCoordinates** object. The coordinate system and the axes do not need to have the same limits. In general, the limits of the coordinate system should be greater than or equal to the limits of the axes. The coordinate system may have limits of 0 to 15, while you may want the axes to extend from 0 to 10.



Graphs can have an UNLIMITED number of x- and y-axes.

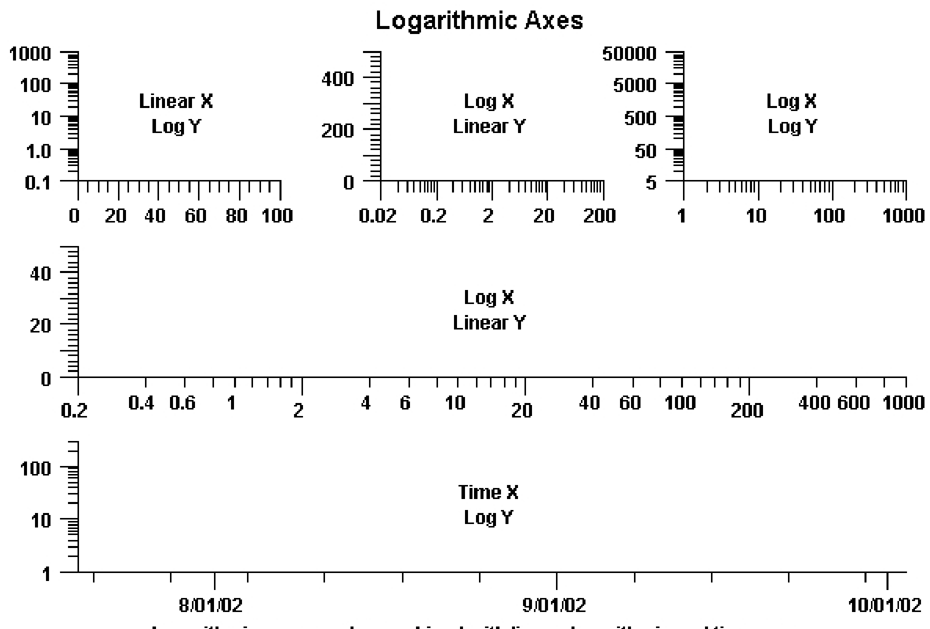
Axis

This class is the abstract base class for the other axis classes. It contains data and drawing routines common to all axis classes.



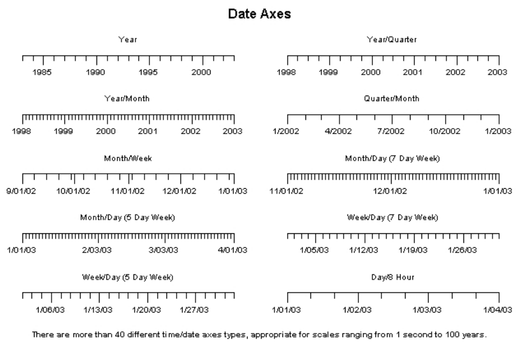
The positioning of axes is very flexible. Axes can have an inverted scale.

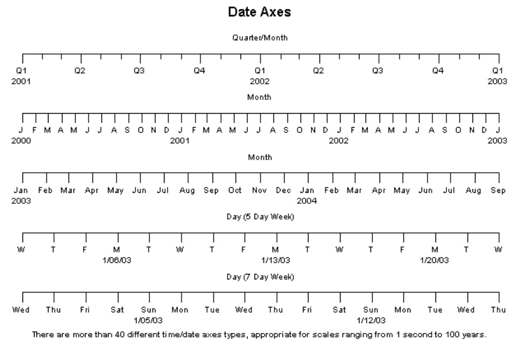
LinearAxis This class implements a linear axis with major and minor tick marks placed at equally spaced intervals.

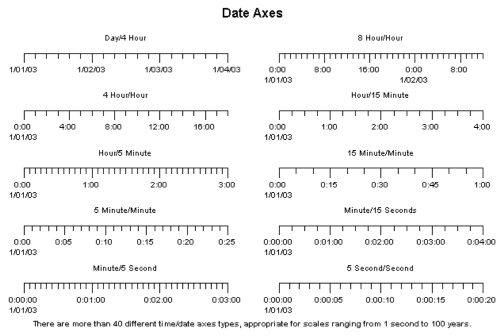


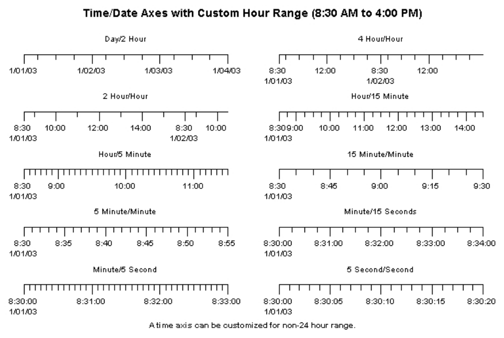
Logarithmic axes can be combined with linear, logarithmic and time axes.

LogAxis This class implements a logarithmic axis with major tick marks placed on logarithmic intervals, for example 1, 10,100 or 30, 300, 3000. The minor tick marks are placed within the major tick marks using linear intervals, for example 2, 3, 4, 5, 6, 7, 8, 9, 20, 30, 40, 50,.., 90. An important feature of the **LogAxis** class is that the major and minor tick marks do not have to fall on decade boundaries. A logarithmic axis must have a positive range exclusive of 0.0, and the tick marks can represent any logarithmic scale.


Date Axes
Year
1985 1990 1995 2000
Year/Quarter
1998 1999 2000 2001 2002 2003
Year/Month
1998 1999 2000 2001 2002 2003
Quarter/Month
1/2002 4/2002 7/2002 10/2002 1/2003
Month/Week
9/01/02 10/01/02 11/01/02 12/01/02 1/01/03
Month/Day (7 Day Week)
11/01/02 12/01/02 1/01/03
Month/Day (5 Day Week)
1/01/03 2/03/03 3/03/03 4/01/03
Week/Day (7 Day Week)
1/05/03 1/12/03 1/19/03 1/26/03
Week/Day (5 Day Week)
1/06/03 1/13/03 1/20/03 1/27/03
Day/8 Hour
1/01/03 1/02/03 1/03/03 1/04/03
There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.

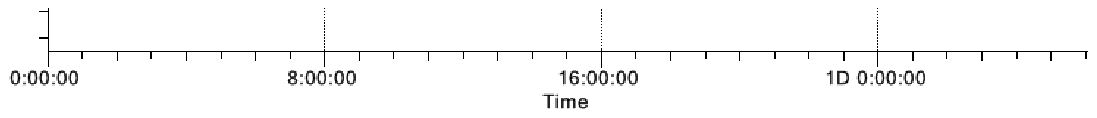

Date Axes
Quarter/Month
Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1
2001 2002 2003
Month
J F M A M J J A S O N D J F M A M J J A S O N D J F M A M J J A S O N D J
2000 2001 2002 2003
Month
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep
2003 2004
Day (5 Day Week)
W T F M T W T F M T W T F M T W
1/06/03 1/13/03 1/20/03
Day (7 Day Week)
Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu
1/05/03 1/12/03
There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.


Date Axes
Day/4 Hour
1/01/03 1/02/03 1/03/03 1/04/03
8 Hour/Hour
0:00 8:00 16:00 0:00 8:00
1/01/03 1/02/03
4 Hour/Hour
0:00 4:00 8:00 12:00 16:00
1/01/03
Hour/15 Minute
0:00 1:00 2:00 3:00 4:00
1/01/03
Hour/5 Minute
0:00 1:00 2:00 3:00
1/01/03
15 Minute/Minute
0:00 0:15 0:30 0:45 1:00
1/01/03
5 Minute/Minute
0:00 0:05 0:10 0:15 0:20 0:25
1/01/03
Minute/15 Seconds
0:00:00 0:01:00 0:02:00 0:03:00 0:04:00
1/01/03
Minute/5 Second
0:00:00 0:01:00 0:02:00 0:03:00
1/01/03
5 Second/Second
0:00:00 0:00:05 0:00:10 0:00:15 0:00:20
1/01/03
There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.


Time/Date Axes with Custom Hour Range (8:30 AM to 4:00 PM)
Day/2 Hour
1/01/03 1/02/03 1/03/03 1/04/03
4 Hour/Hour
8:30 12:00 8:30 12:00
1/01/03 1/02/03
2 Hour/Hour
8:30 10:00 12:00 14:00 8:30 10:00
1/01/03 1/02/03
Hour/15 Minute
8:30 9:00 10:00 11:00 12:00 13:00 14:00
1/01/03
Hour/5 Minute
8:30 9:00 10:00 11:00
1/01/03
15 Minute/Minute
8:30 8:45 9:00 9:15 9:30
1/01/03
5 Minute/Minute
8:30 8:35 8:40 8:45 8:50 8:55
1/01/03
Minute/15 Seconds
8:30:00 8:31:00 8:32:00 8:33:00 8:34:00
1/01/03
Minute/5 Second
8:30:00 8:31:00 8:32:00 8:33:00
1/01/03
5 Second/Second
8:30:00 8:30:05 8:30:10 8:30:15 8:30:20
1/01/03
A time axis can be customized for non-24 hour range.

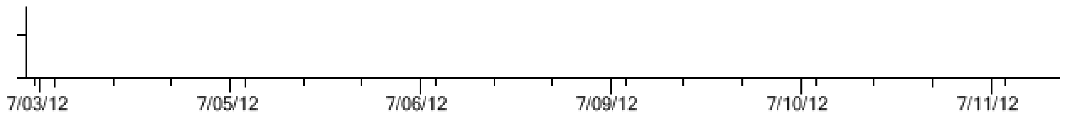
TimeAxis

This class is the most complex of the axis classes. It supports time scales ranging from 1 millisecond to hundreds of years. Dates and times are specified using the **GregorianCalendar** class. The major and minor tick marks can fall on any time base, where a time base represents seconds, minutes, hours, days, weeks, months or years. The scale can exclude weekends, for example, Friday, October 20, 2000 is immediately followed by Monday, October 23, 2000. A day can also have a custom range, for example a range of 9:30 AM to 4:00 PM. The chart time axis excludes time outside of this range. This makes the class very useful for the inter-day display of financial market information (stock, bonds, commodities, options, etc.) across several days, months or years.



ElapsedTimeAxis

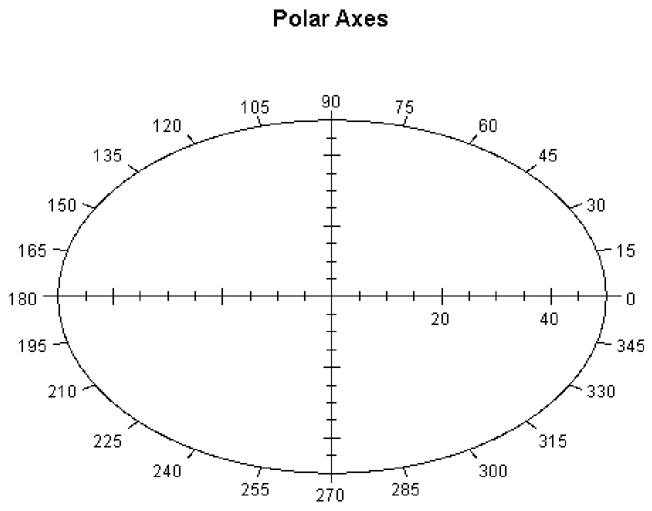
The elapsed time axis is very similar to the linear axis and is subclassed from that class. The main difference is the major and minor tick mark spacing calculated by the CalcAutoAxis method takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. It is a continuous linear scale.



EventAxis

The event axis is a hybrid of the a time axis and the linear axis. It places major and minor tick marks on the axis, based on the event data attached to the coordinate system. Every ChartEvent object in the coordinate system does not necessarily have a tick mark, because where the data values are bunched together, this would create too many tick marks and they would overlap. Every tick mark, though, will have at least one ChartEvent object associated with it.

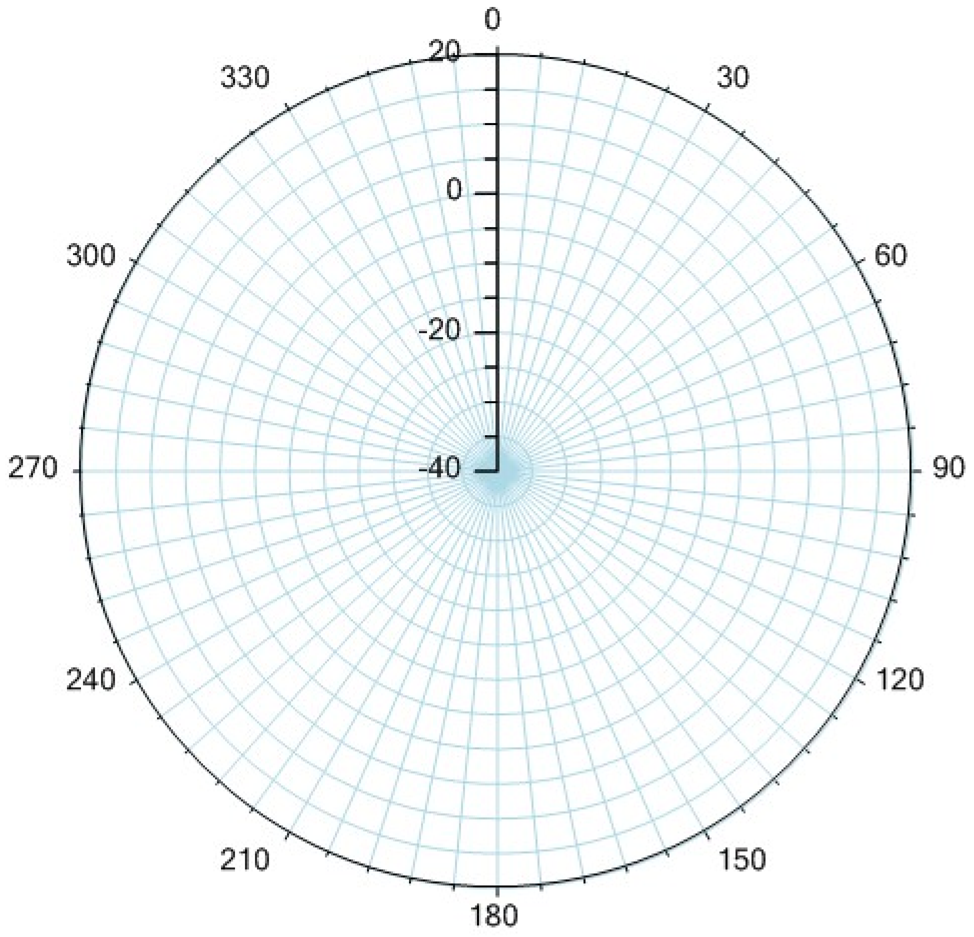
In the case where multiple plots have ChartEvent objects with the same time stamp, a tick mark can have multiple ChartEvent objects associated with it.



A polar axis consists of the x and y axis for magnitude, and the outer circle for the angle.

PolarAxes

This class has three separate axes: two linear and one circular. The two linear axes, scaled for +- the magnitude of the polar scale, form a cross with the center of both axes at the origin (0, 0. The third axis is a circle centered on the origin with a radius equal to the magnitude of the polar scale. This circular axis represents 360 degrees (or 2 Pi radians) of the polar scale and the tick marks that circle this axis are spaced at equal degree intervals.



AntennaAxes

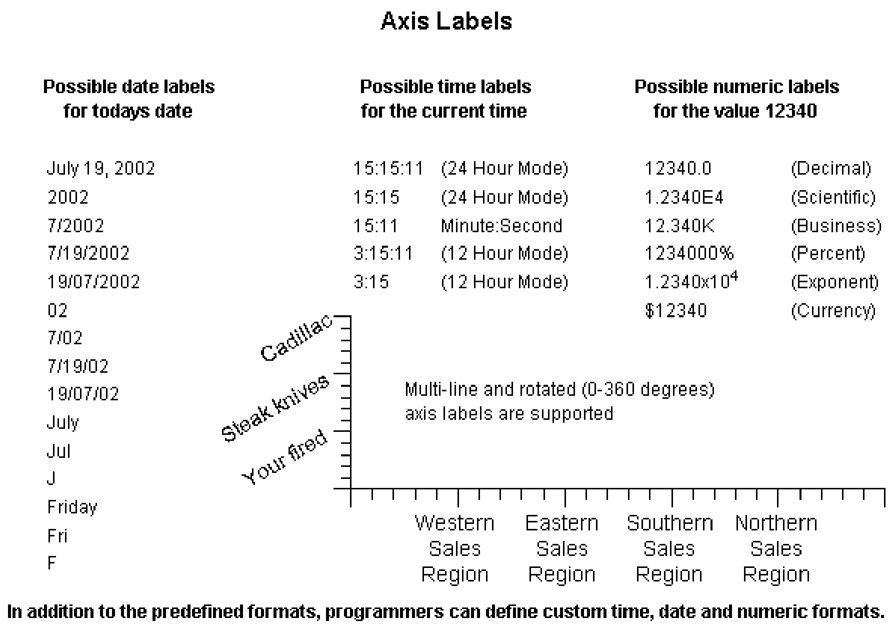
This class has two axes: one linear y-axis and one circular axis. The linear axis is scaled for the desired range of radius values. This can extend from minus values to plus values. The second axis is a circle centered on the origin with a radius equal to the range of the radius scale. This circular axis represents 360 degrees of the antenna scale and the tick marks that circle this axis are spaced at equal degree intervals.

Axis Label Classes

AxisLabels
- **NumericAxisLabels**
- **StringAxisLabels**
- **PolarAxesLabels**

AntennaAxesLabels
TimeAxisLabels
ElapsedTimeAxisLabels
EventAxisLabels

Axis labels inform the user of the x- and y-scales used in the chart. The labels center on the major tick marks of the associated axis. Axis labels are usually numbers, times, dates, or arbitrary strings.



AxisLabels	This class is the abstract base class for all axis label objects. It places numeric labels, date/time labels, or arbitrary text labels, at the major tick marks of the associated axis object. In addition to the standard font options (type, size, style, color, etc.), axis label text can be rotated 360 degrees in one degree increments.
NumericAxisLabels	This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes. The class supports many predefined and user-definable formats, including numeric, exponent, percentage, business and currency formats.
StringAxisLabels	This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes using user-defined strings.

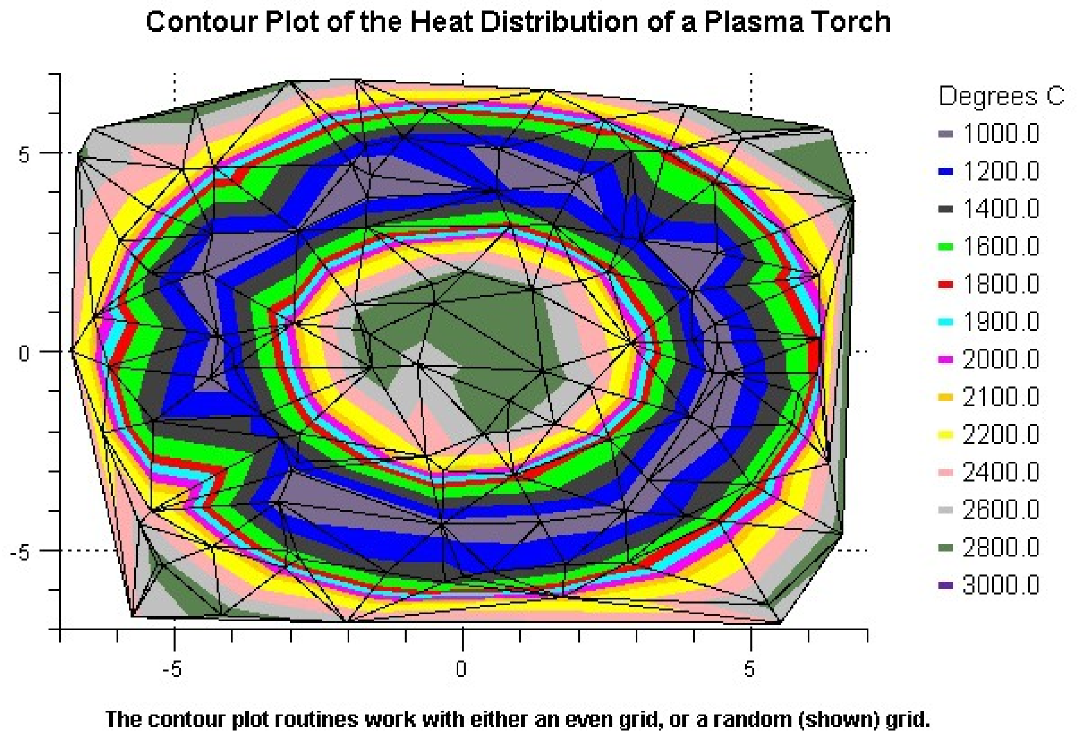
TimeAxisLabels	This class labels the major tick marks of the associated **TimeAxis** object. The class supports many time (23:59:59) and date (5/17/2001) formats. It is also possible to define custom date/time formats.
ElapsedTimeAxisLabels	This class labels the major tick marks of the associated **ElapsedTimeAxis** object. The class supports HH:MM:SS and MM:SS formats, with decimal seconds out to 0.00001, i.e. “12:22:43.01234”. It also supports a cumulative hour format (101:51:22), and a couple of day formats (4.5:51:22, 4D 5:51:22).
PolarAxesLabels	This class labels the major tick marks of the associated **PolarAxes** object. The x-axis is labeled from 0.0 to the polar scale magnitude, and the circular axis is labeled counter clockwise from 0 to 360 degrees, starting at 3:00.
AntennaAxesLabels	This class labels the major tick marks of the associated **AntennaAxes** object. The y-axis is labeled from the radius minimum to the radius maximum. The circular axis is labeled clockwise from 0 to 360 degrees, starting at 12:00.
EventAxisLabels	This class labels the major tick marks of the associated **EventAxis** object. The class supports many time (23:59:59) and date (5/17/2001) formats. It is also possible to define custom date/time formats.

Chart Plot Classes

ChartPlot
- **ContourPlot**
- **GroupPlot**
- **PieChart**
- **PolarPlot**
- **SimplePlot**

Plot objects are objects that display data organized in a **ChartDataset** class. There are five main categories: simple, group, polar, contour and pie plots. Simple plots graph data organized as a simple set of xy data points. The most common examples of simple plots are line plots, bar graphs and scatter plots. Group plots graph data organized as multiple y-values for each x-value. The most common examples of group plots are stacked bar graphs, open-high-low-close plots, and candlestick plots. Polar charts plot data organized as a simple set of data points, where each data point represents a polar magnitude and angle pair, rather than xy Cartesian coordinate values. The most common example of polar charts is the display of complex numbers (a + b*i*), and it is used in many engineering disciplines. The contour plot type displays the iso-lines, or contours, of a 3D surface using either lines or regions of solid color. The last plot object category is the pie chart, were a pie wedge represents each data value. The size of the pie wedge is proportional to the fraction (data value / sum of all data values).

ChartPlot This class is the abstract base class for chart plot objects. It contains a reference to a **ChartDataset** derived class containing the data associated with the plot.



ContourPlot This class is a concrete implementation of the **ChartPlot** class and displays a contour plot using either lines, or regions filled with color.

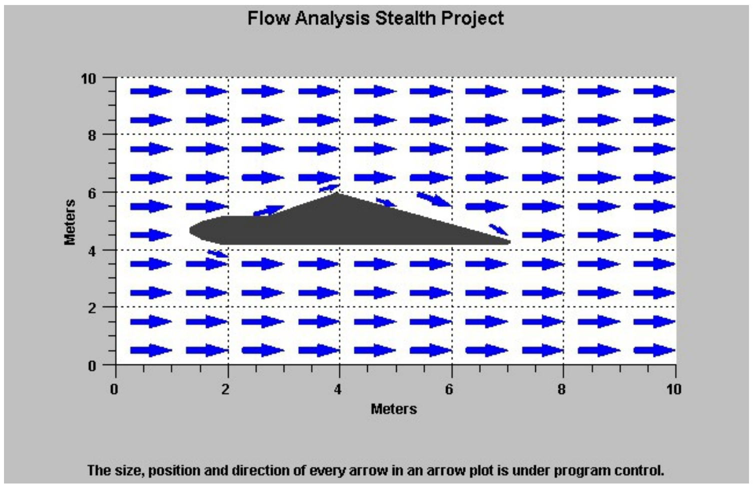
Group Plot Classes

GroupPlot

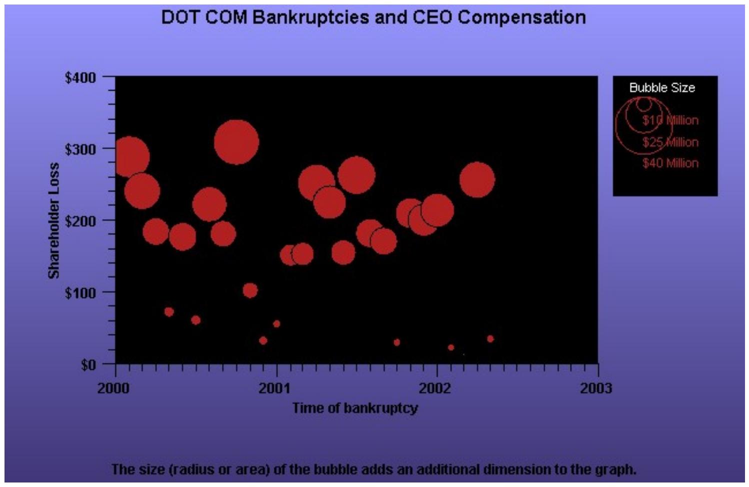
- **ArrowPlot**
- **BubblePlot**
- **CandlestickPlot**
- **CellPlot**
- **ErrorBarPlot**
- **FloatingBarPlot**
- **GroupBarPlot**
- **HistogramPlot**
- **LineGapPlot**
- **MultiLinePlot**
- **OHLCPlot**
- **StackedBarPlot**
- **StackedLinePlot**

Group plots use data organized as arrays of x- and y-values, where there is one or more y for every x.. Group plot types include multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots, open-high-low-close plots, candlestick plots, arrow plots, histogram plots, cell plots and bubble plots.

GroupPlot This class is an abstract base class for all group plot classes.

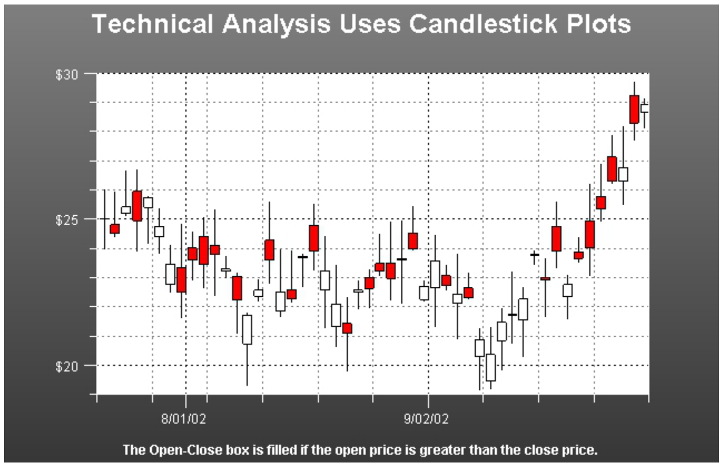


ArrowPlot This class is a concrete implementation of the **GroupPlot** class and it displays a collection of arrows as defined by the data in a group dataset. The position, size, and rotation of each arrow in the collection is independently controlled



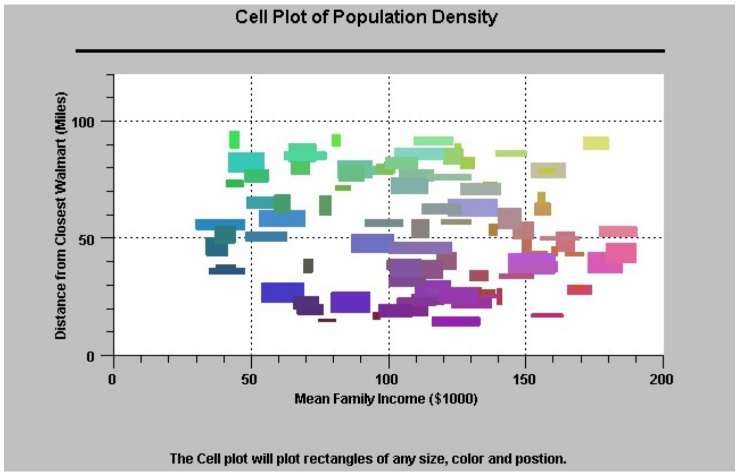
BubblePlot

This class is a concrete implementation of the **GroupPlot** class and displays bubble plots. The values in the dataset specify the position and size of each bubble in a bubble chart.



CandlestickPlot

This class is a concrete implementation of the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis.

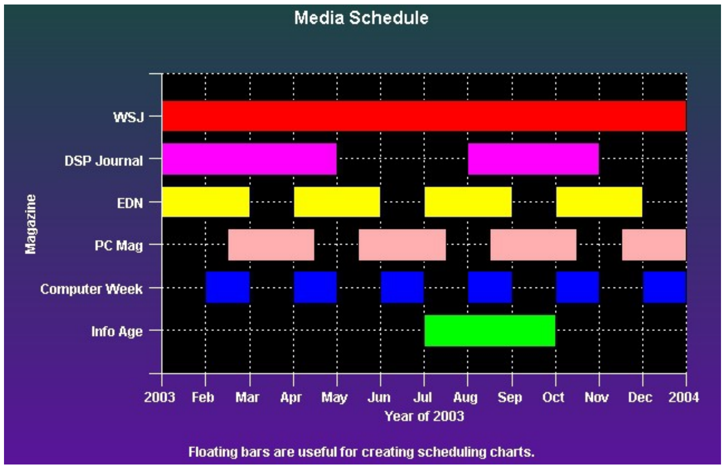


CellPlot

This class is a concrete implementation of the **GroupPlot** class and displays cell plots. A cell plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset.

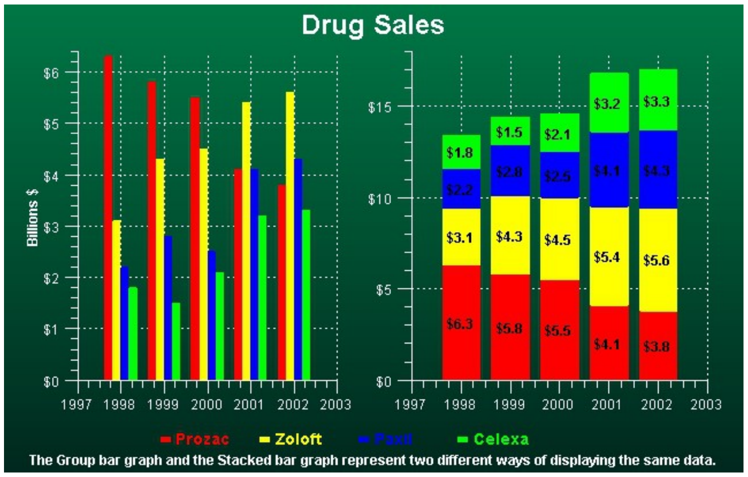
ErrorBarPlot

This class is a concrete implementation of the **GroupPlot** class and displays error bars. Error bars are two lines positioned about a data point that signify the statistical error associated with the data point



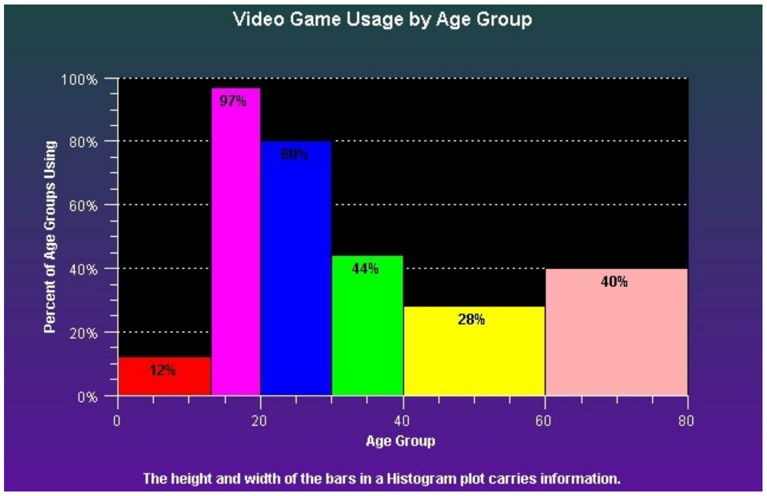
FloatingBarPlot

This class is a concrete implementation of the **GroupPlot** class and displays free-floating bars in a graph. The bars are free floating because each bar does not reference a fixed base value, as do simple bar plots, stacked bar plots and group bar plots.

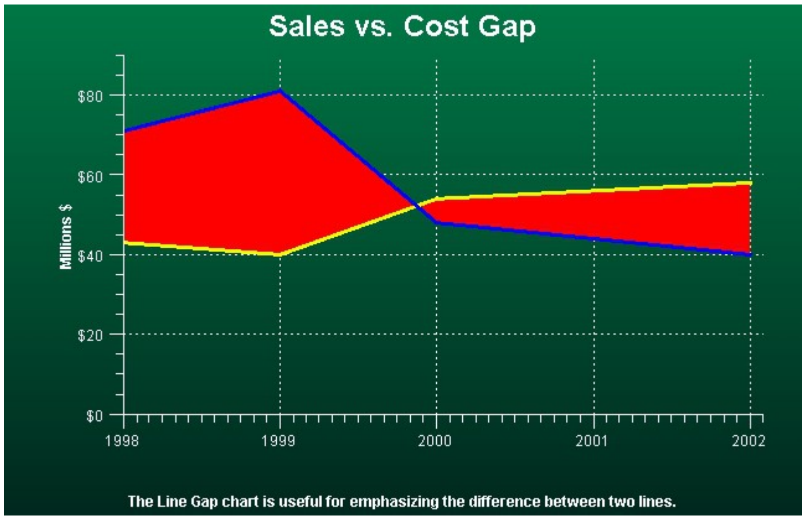


GroupBarPlot This class is a concrete implementation of the **GroupPlot** class and displays group data in a group bar format. Individual bars, the height of which corresponds to the group y-values of the dataset, display side by side, as a group, justified with respect to the x-position value for each group. The group bars share a common base value.

StackedBarPlot This class is a concrete implementation of the **GroupPlot** class and displays data as stacked bars. In a stacked bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it.

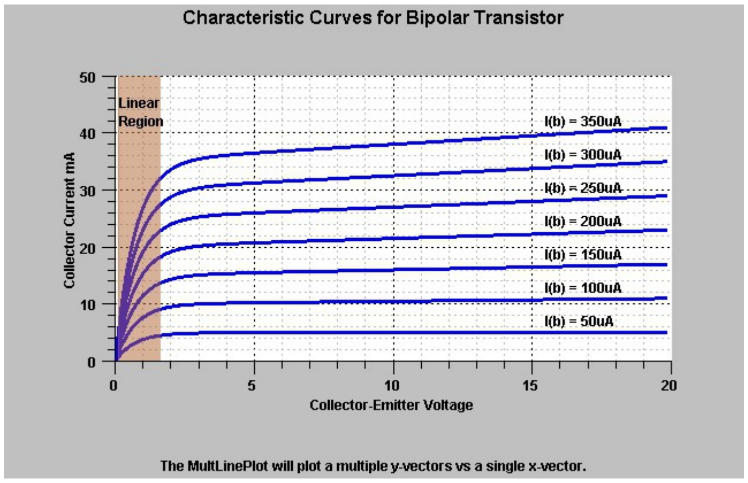


HistogramPlot This class is a concrete implementation of the **GroupPlot** class and displays histogram plots. A histogram plot is a collection of rectangular objects with independent widths and heights, specified using the values of the associated group dataset. The histogram bars share a common base value.



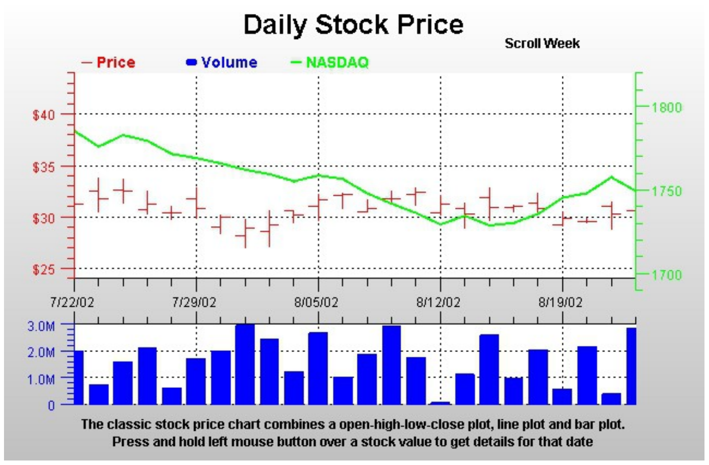
LineGapPlot

This class is a concrete implementation of the **GroupPlot** class. A line gap chart consists of two lines plots where a contrasting color fills the area between the two lines, highlighting the difference.



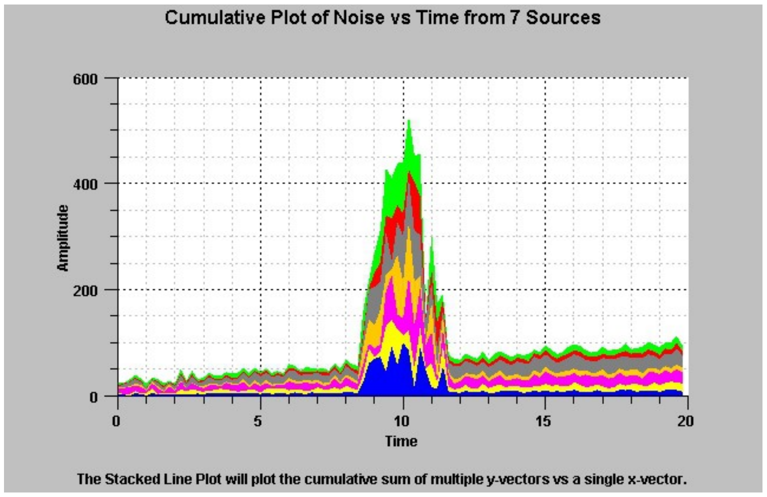
MultiLinePlot

This class is a concrete implementation of the **GroupPlot** class and displays group data in multi-line format. A group dataset with four groups will display four separate line plots. The y-values for each line of the line plot represent the y-values for each group of the group dataset. Each line plot share the same x-values of the group dataset.



OHLCPlot

This class is a concrete implementation of the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is a vertical line, representing High and Low values, with two small horizontal "flags", one left and one right extending from the vertical High-Low line and representing the Open and Close values.



StackedLinePlot

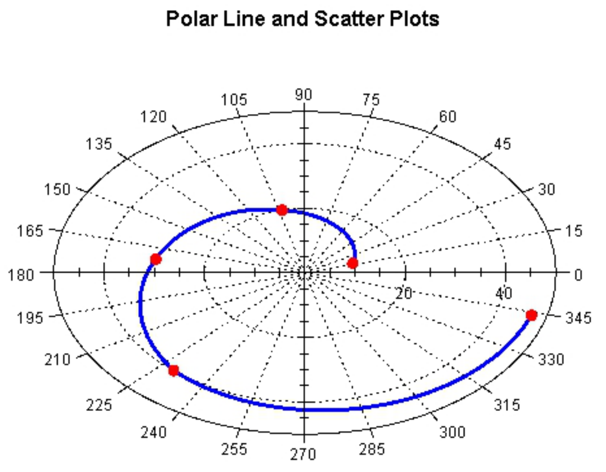
This class is a concrete implementation of the **GroupPlot** class and displays data in a stacked line format. In a stacked line plot each group is stacked on top of one another, each group line a cumulative sum of the related group items before it.

Polar Plot Classes

PolarPlot
- **PolarLinePlot**
- **PolarScatterPlot**

Polar plots that use data organized as arrays of x- and y-values, where an x-value represents the magnitude of a point in polar coordinates, and the y-value represents the angle, in radians, of a point in polar coordinates. Polar plot types include line plots and scatter plots.

PolarPlot	This class is an abstract base class for the polar plot classes.

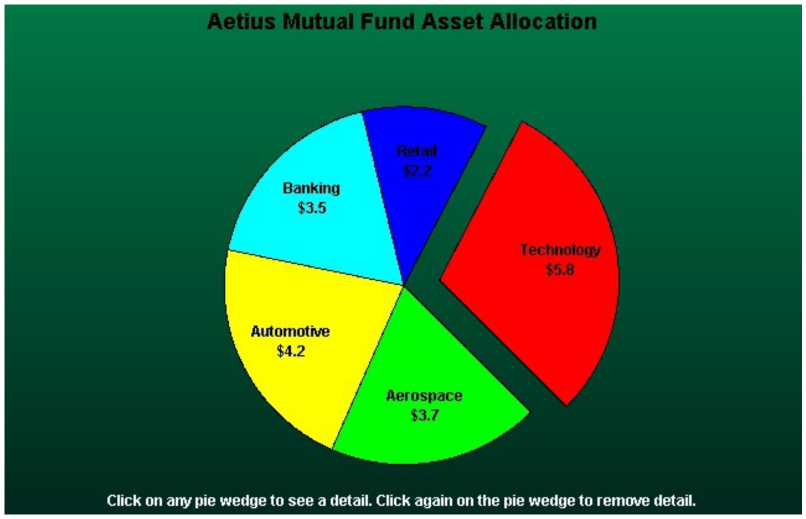


The polar line charts use true polar (not linear) interpolation between data points.

PolarLinePlot	This class is a concrete implementation of the **PolarPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use polar coordinate interpolation.
PolarScatterPlot	This class is a concrete implementation of the **PolarPlot** class and displays data in a simple scatter plot format.

Pie Chart Classes

It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a pie wedge, and a y-value specifies the offset (or "explosion") of a pie wedge with respect to the center of the pie.



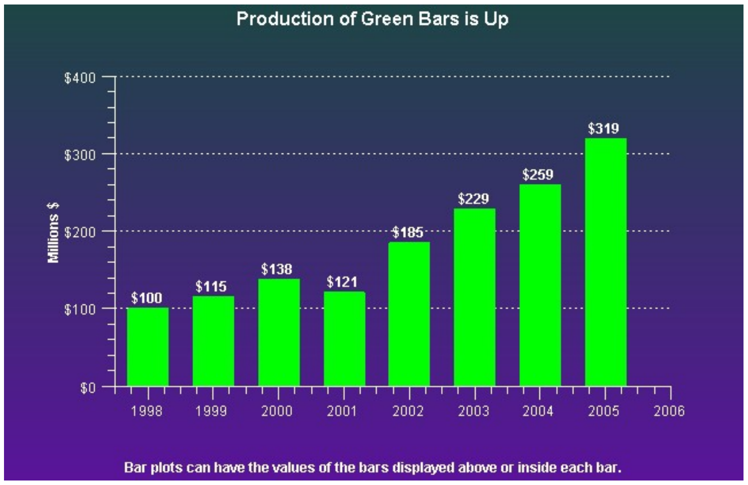
PieChart	This class plots data in a simple pie chart format. It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a pie wedge, and a y-value specifies the offset (or "explosion") of a pie wedge with respect to the center of the pie.

Simple Plot Classes

SimplePlot
- **SimpleBarPlot**
- **SimpleLineMarkerPlot**
- **SimpleLinePlot**
- **SimpleScatterPlot**

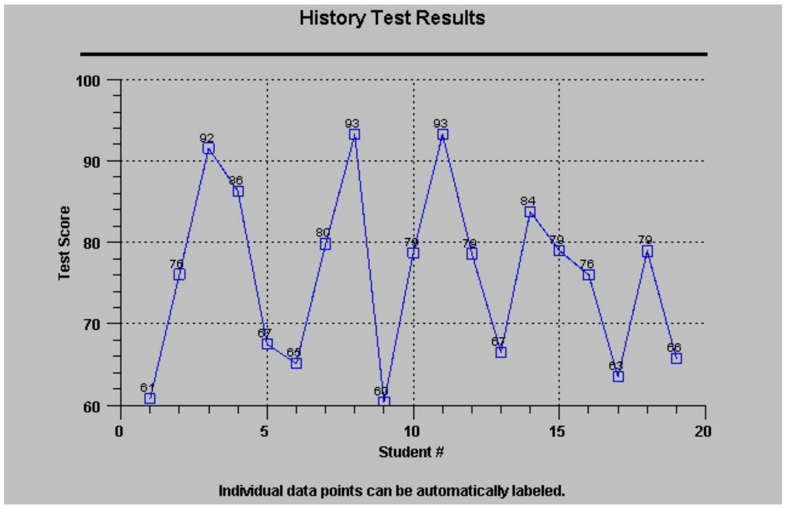
Simple plots use data organized as a simple array of xy points, where there is one y for every x. Simple plot types include line plots, scatter plots, bar graphs, and line-marker plots.

SimplePlot	This class is an abstract base class for all simple plot classes.



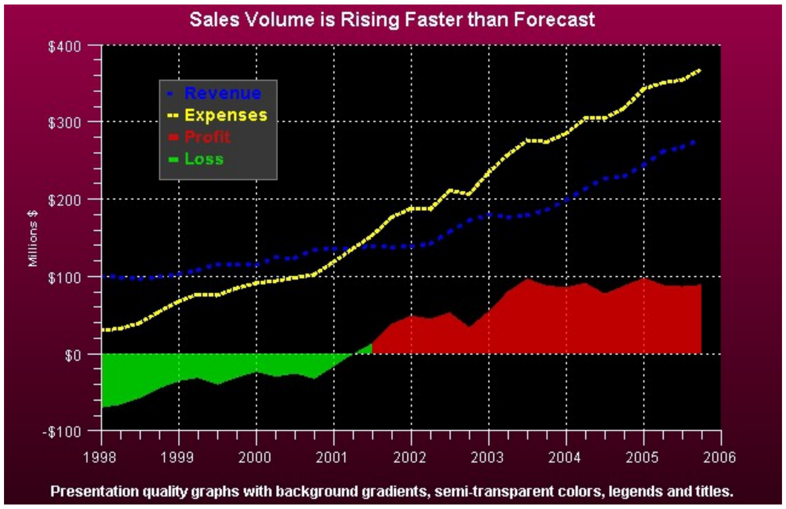
SimpleBarPlot

This class is a concrete implementation of the **SimplePlot** class and displays data in a bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, are justified with respect to the x-values.

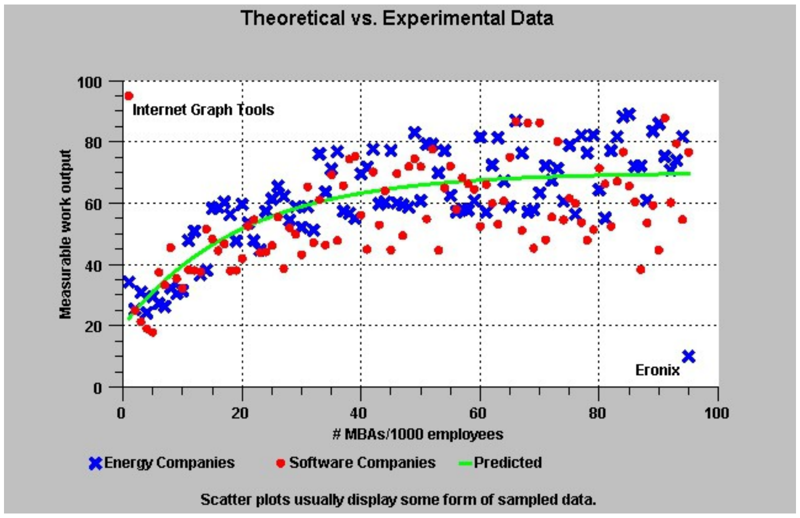


SimpleLineMarkerPlot

This class is a concrete implementation of the **SimplePlot** class and it displays simple datasets in a line plot format where scatter plot symbols highlight individual data points.



SimpleLinePlot This class is a concrete implementation of the **SimplePlot** class it displays simple datasets in a line plot format. Adjacent data points are connected using a straight, or a step line.



SimpleScatterPlot This class is a concrete implementation of the **SimplePlot** class and it displays simple datasets in a scatter plot format where each data point is represented using a symbol.

Legend Classes

LegendItem
BubblePlotLegendItem
Legend

StandardLegend
BubblePlotLegend

Legends provide a key for interpreting the various plot objects in a graph. It organizes a collection of legend items, one for each plot objects in the graph, and displays them in a rectangular frame.

Legend	This class is the abstract base class for chart legends.
LegendItem	This class is the legend item class for all plot objects except for bubble plots. Each legend item manages one symbol and descriptive text for that symbol. The **StandardLegend** class uses objects of this type as legend items.
BubblePlotLegendItem	This class is the legend item class for bubble plots. Each legend item manages a circle and descriptive text specifying the value of a bubble of this size. The **BubblePlotLegend** class uses objects of this type as legend items.
StandardLegend	This class is a concrete implementation of the **Legend** class and it is the legend class for all plot objects except for bubble plots. The legend item objects display in a row or column format. Each legend item contains a symbol and a descriptive string. The symbol normally associates the legend item to a particular plot object, and the descriptive string describes what the plot object represents.
BubblePlotLegend	This class is a concrete implementation of the **Legend** class and it is a legend class used exclusively with bubble plots. The legend item objects display as offset, concentric circles with descriptive text giving the key for the value associated with a bubble of this size.

Grid Classes

Grid
PolarGrid

Grid lines are perpendicular to an axis, extending the major and/or minor tick marks of the axis across the width or height of the plot area of the chart.

Grid	This class defines the grid lines associated with an axis. Grid lines are perpendicular to an axis, extending the major and/or minor tick marks of the axis across the width or height of the plot area of the chart. This class works in conjunction with the **LinearAxis**, **LogAxis** and **TimeAxis** classes.
PolarGrid	This class defines the grid lines associated with a polar axis. A polar chart grid consists of two sets of lines. The first set is a group of concentric circles, centered on the origin and passing through the major and/or minor tick marks of the polar magnitude horizontal and vertical axes. The second set is a group of radial lines, starting at the origin and extending to the outermost edge of the polar plot circle, passing through the major and minor tick marks of the polar angle circular axis. This class works in conjunction with the **PolarAxes** class.

Chart Text Classes

- **ChartText**
  - **ChartTitle**
  - **AxisTitle**
  - **ChartLabel**
    - **NumericLabel**
    - **TimeLabel**
    - **StringLabel**

The chart text classes draw one or more strings in the chart window. Different classes support different numeric formats, including floating-point numbers, date/time values and multi-line text strings. International formats for floating-point numbers and date/time values are also supported.

ChartText	This class draws a string in the current chart window. It is the base class for the **ChartTitle**, **AxisTitle** and **ChartLabel** classes. The **ChartText** class also creates independent text objects. Other classes that display text also use it internally.

ChartTitle	This class displays a text string as the title or footer of the chart.
AxisTitle	This class displays a text string as the title for an axis. The axis title position is outside of the axis label area. Axis titles for y-axes are rotated 90 degrees.
ChartLabel	This class is the abstract base class of labels that require special formatting.
NumericLabel	This class is a concrete implementation of the **ChartLabel** class and it displays formatted numeric values.
TimeLabel	This class is a concrete implementation of the **ChartLabel** class and it displays formatted **GregorianCalendar** dates.
StringLabel	This class is a concrete implementation of the **ChartLabel** class that formats string values for use as axis labels.

Miscellaneous Chart Classes

Marker
ChartImage
ChartShape
ChartSymbol

Various classes are used to position and draw objects that can be used as standalone objects in a graph, or as elements of other plot objects.

Marker	This class displays one of five marker types in a graph. The marker is used to create data cursors, or to mark data points.
ChartImage	This class encapsulates a **Java Image** class, defining a rectangle in chart coordinates that the image is placed in. JPEG and other image files can be imported using the Java **Image** class and displayed in a chart.
ChartShape	This class encapsulates a **Java Shape** class, placing the shape in a chart using a position defined in chart

	coordinates. A chart can display any object that can be defined using **Shape** class.
ChartSymbol	This class defines symbols used by the **SimplePlot** scatter plot functions. Pre-defined symbols include square, triangle, diamond, cross, plus, star, line, horizontal bar, vertical bar, 3D bar and circle.

Mouse Interaction Classes

ChartMouseListener
- **MoveObj**
- **FindObj**
- **DataToolTip**
- **ChartZoom**

Marker
- **DataCursor**
  - **MoveData**

Several classes implement Swing **MouseInputListener** interface class. The **ChartMouseListener** class implements a generic interface for managing mouse events in a graph window. The **DataCursor**, **MoveData**, **MoveObj** and **ChartZoom** classes also implement the **MouseInputListener** interface, using the mouse to mark, move and zoom chart objects and data.

ChartMouseListener	This class implements the Java **javax.swing.event.MouseInputListener** interface. It traps generic mouse events (button events and mouse motion events) that take place in a **ChartView** window. A programmer can derive a class from **ChartMouseListener** and override the methods for mouse events, creating a custom version of the class.
MoveObj	This class extends the **ChartMouseListener** class and it can select chart objects and move them. Moveable chart objects include axes, axes labels, titles, legends, arbitrary text, shapes and images. Use the **MoveData** class to move objects derived from **SimplePlot**.
FindObj	This class extends the **ChartMouseListener** class, providing additional methods that selectively determine what graphical objects intersect the mouse cursor.

DataCursor	This class combines the **MouseInputListener** interface and **Marker** class. Press a mouse button and the selected data cursor (horizontal and/or vertical line, cross hairs, or a small box) appears at the point of the mouse cursor. The data cursor tracks the mouse motion as long as the mouse button is pressed. Release the button and the data cursor disappears. This makes it easier to line up the mouse position with the tick marks of an axis.
MoveData	This class selects and moves individual data points of an object derived from the **SimplePlot** class.
DataToolTip	A data tooltip is a popup box that displays the value of a data point in a chart. The data value can consist of the x-value, the y-value, x- and y-values, group values and open-high-low-close values, for a given point in a chart.
ChartZoom	This class implements mouse controlled zooming for one or more simultaneous axes. The user starts zooming by holding down a mouse button with the mouse cursor in the plot area of a graph. The mouse is dragged and then released. The rectangle established by mouse start and stop points defines the new, zoomed, scale of the associated axes. Zooming has many different modes. Some of the combinations are:

⑤ One x or one y axis
⑤ One x and one y axes
⑤ One x and multiple y axes
⑤ One y and multiple x axes
⑤ Multiple x and y axes

File and Printer Rendering Classes

ChartPrint
ChartBufferedImage

ChartPrint	This class implements the Java **Printable** interface. It can select, setup, and output a chart to a printer.
ChartBufferedImage	This class will convert an **ChartView** object to a Java **BufferedImage** object. Optionally, the class saves the buffered image to a JPEG file.

Miscellaneous Utility Classes

ChartConstants
ChartCalendar
CSV
ChartDimension
ChartPoint2D
GroupPoint2D
DoubleArray
DoubleArray2D
BoolArray
ChartPoint3D
NearestPointData
TickMark
Polysurface
ChartRectangle2D

ChartConstants This interface consolidates all of the constant values used in the com.quinncurtis.chart2djava package**.**

ChartCalendar	This class contains utility routines used to process **GregorianCalendar** date objects.
CSV	This is a utility class for reading and writing CSV (Comma Separated Values) files.
ChartDimension	This is a utility class for handling dimension (height and width) information using doubles, rather than the integers used by the Size class.
ChartPoint2D	This class encapsulates an xy pair of values as doubles.
GroupPoint2D	This class encapsulates an x-value, and an array of y-values, representing the x and y values of one column of a group data set.
DoubleArray	This class is used as an alternative to the standard Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.
DoubleArray2D	This class is used as an alternative to the standard Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.

BoolArray	This class is used as an alternative to the standard Java Array class, adding routines for resizing of the array, and the insertion and deletion of bool based data elements.
ChartPoint3D	This class encapsulates an xyz set of double values used to specify 3D data values.
NearestPointData	This is a utility class for returning data that results from nearest point calculations.
TickMark	The axis classes use this class to organize the location of the individual tick marks of an axis.
Polysurface	This is a utility class that defines complex 3D shapes as a list of simple 3-sided polygons. The contour plotting routines use it.
ChartRectangle2D	This is a utility class that extends the Rectangle2D.Double class.

A diagram depicts the class hierarchy of the QCChart2D for Java library.

- ChartConstants $\longrightarrow$
  - ChartObj
    - Arrow
    - GregorianCalendar
    - CSV
    - ChartDimension
    - ChartPoint3D
    - GroupPoint2D
    - NearestPointData
    - Polysurface
    - ChartScale
      - LinearScale
      - LogScale
      - TimeScale
    - UserCoordinates
      - WorldCoordinates
        - WorkingCoordinates
          - PhysicalCoordinates
            - CartesianCoordinates
              - PolarCoordinates
            - TimeCoordinates
    - ChartDataset
      - SimpleDataset
        - TimeSimpleDataset
        - ContourDataset
      - GroupDataset
        - TimeGroupDataset
    - AutoScale
      - LinearAutoScale
      - LogAutoScale
      - TimeAutoScale
    - ChartMouseListener
      - MoveObj
      - FindObj
      - DataToolTip
      - ChartZoom
  - DataCursor
    - MoveData
  - ChartAttribute
  - ChartGradient
  - ChartPrint
  - ChartBufferedImage

- javax.swing.JPanel
  - ChartView

- Rectangle2D.Double
  - ChartRectangle2D
- Point2D.Double
  - ChartPoint2D
- DoubleArray
- DoubleArray2D
- BoolArray
- Polysurface

- GraphObj
  - TickMark
  - Axis
    - LinearAxis
      - PolarAxes
    - LogAxis
    - TimeAxis
  - ChartText
    - ChartTitle
    - AxisTitle
    - ChartLabel
      - NumericLabel
        - BarDatapointValue
      - TimeLabel
      - StringLabel
    - AxisLabels
      - NumericAxisLabels
      - TimeAxisLabels
      - StringAxisLabels
      - PolarAxesLabels
  - Grid
    - PolarGrid
  - LegendItem
  - BubblePlotLegendItem
  - Legend
    - StandardLegend
    - BubblePlotLegend
  - ChartPlot
    - SimplePlot
      - SimpleLinePlot
      - SimpleBarPlot
      - SimpleScatterPlot
      - SimpleLineMarkerPlot
    - GroupPlot
      - ArrowPlot
      - BubblePlot
      - CandlestickPlot
      - CellPlot
      - ErrorBarPlot
      - FloatingBarPlot
      - GroupBarPlot
      - HistogramPlot
      - LineGapPlot
      - MultiLinePlot
      - OHLCPlot
      - StackedBarPlot
      - StackedLinePlot
    - PieChart
    - PolarPlot
      - PolarLinePlot
      - PolarScatterPlot
  - Background
  - ChartImage
  - ChartShape
  - ChartSymbol
  - Marker
  - ChartZoom

5. SPC Control Data and Alarm Classes

SPCControlChartData
SPCControlLimitAlarmArgs
SPCControlLimitRecord
SPCCalculatedValueRecrod
SPCSampledValueRecord
SPCGeneralizedTableDisplay

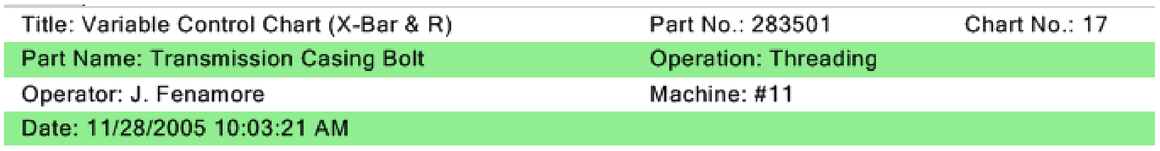
The *Variable* and *Attribute Control Chart* classes share common data and alarm classes. SPC control chart data is stored in the **SPCControlChartData** class. It holds the header information used to customize the chart table, the raw sample data used to prepare the chart, the calculated chart values used in the chart, and the SPC control limits. It contains array lists of **SPCSampledValueRecord**, **SPCControlLimitRecord** and **SPCCalculatedValueRecord** objects. The **SPCGeneralizedTableDisplay** class manages **ChartText** objects used to display data in the table portion of the SPC chart.

Class SPCControlChartData

```
ChartObj
        |
        +-- SPCControlChartData
```

The **SPCControlChartData** class is the core data storage object for all of SPC Control Chart classes. It holds all of the data plotted in the SPC chart. That includes the header information used to customize the chart table,

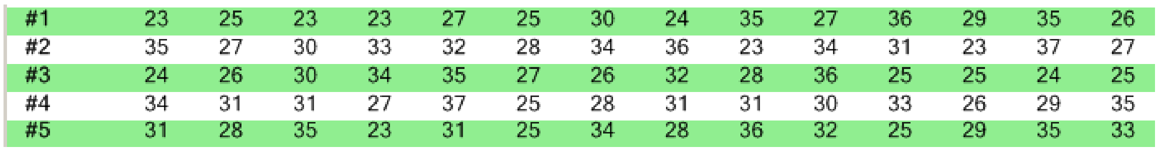
Header Information



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17
Part Name: Transmission Casing Bolt	Operation: Threading	
Operator: J. Fenamore	Machine: #11	
Date: 11/28/2005 10:03:21 AM		

the raw sample data used in the SPC calculations,

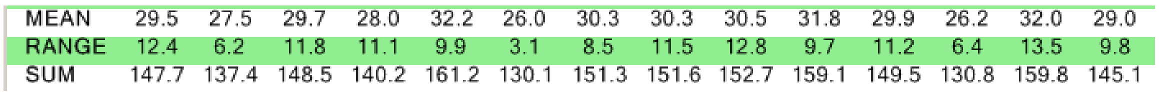
Raw Sample Data



#1	23	25	23	23	27	25	30	24	35	27	36	29	35	26
#2	35	27	30	33	32	28	34	36	23	34	31	23	37	27
#3	24	26	30	34	35	27	26	32	28	36	25	25	24	25
#4	34	31	31	27	37	25	28	31	31	30	33	26	29	35
#5	31	28	35	23	31	25	34	28	36	32	25	29	35	33

the calculated chart values used in the chart, and the SPC control limits,

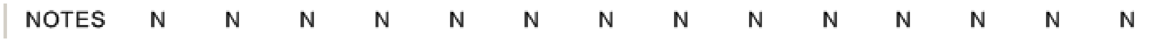
Calculated Values



MEAN	29.5	27.5	29.7	28.0	32.2	26.0	30.3	30.3	30.5	31.8	29.9	26.2	32.0	29.0
RANGE	12.4	6.2	11.8	11.1	9.9	3.1	8.5	11.5	12.8	9.7	11.2	6.4	13.5	9.8
SUM	147.7	137.4	148.5	140.2	161.2	130.1	151.3	151.6	152.7	159.1	149.5	130.8	159.8	145.1

and any notes you might want to place in the record.

Notes



NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N

It contains array lists of **SPCSampledValueRecord**, **SPCControlLimitRecord** and **SPCCalculatedValueRecord** objects.

There is an instance of **SPCControlChartData** in the **SPCChartBase** class. Since the **SPCChartBase** class is the base class for the four major SPC Control Charts (**SPCBatchAttributeConrolChart**, **SPCBatchVariableControlChart**, **SPCTimeAttributeConrolChart**, **SPCTimeVariableControlChart**), it is accessible from those classes. The data elements of the **SPCControlChartData** class are accessible to the programmer.

SPCControlChartData Methods

The **SPCControlChartData** object is automatically created when the parent **SPCChartBase** object is created. The programmer does not need to instantiate it.

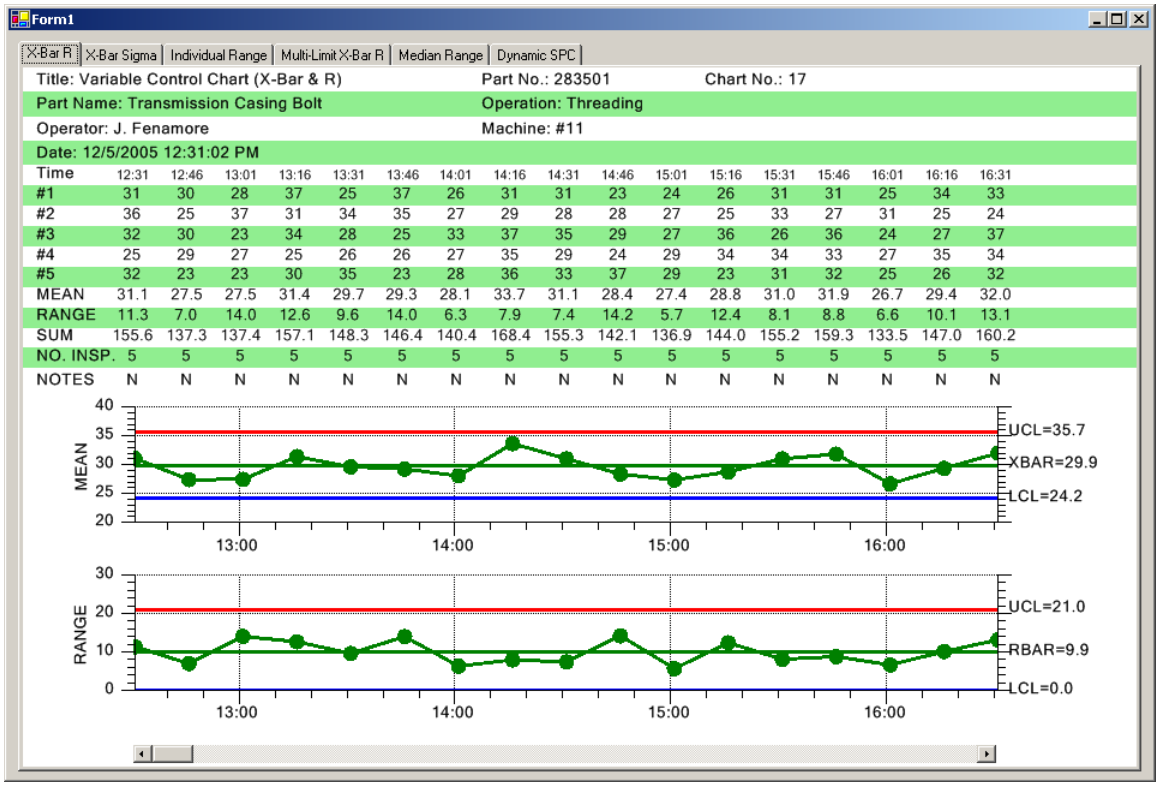
Class SPCControlChartData – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCControlChartData.htm

a link to the JavaDoc documentation for the class **SPCControlChartData**

Initializing the SPCControlChartData Class

The control charts **initSPC**… method call initializes the **SPCControlChartData** object. This establishes the SPC chart type, how many samples per subgroup there are, and how many **SPCSampledValueRecord** objects are stored internal to the **SPCControlChartData** to handle the sampled data.

The table strings used to customize the first section of the chart should be set after the chart **initSPC**… call, but before the **rebuildChartUsingCurrentData** call. The example below is from the **TimeVariableControlCharts.XBarRChart** example program.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 12/5/2005 12:31:02 PM

Time	12:31	12:46	13:01	13:16	13:31	13:46	14:01	14:16	14:31	14:46	15:01	15:16	15:31	15:46	16:01	16:16	16:31
#1	31	30	28	37	25	37	26	31	31	23	24	26	31	31	25	34	33
#2	36	25	37	31	34	35	27	29	28	28	27	25	33	27	31	25	24
#3	32	30	23	34	28	25	33	37	35	29	27	36	26	36	24	27	37
#4	25	29	27	25	26	26	27	35	29	24	29	34	34	33	27	35	34
#5	32	23	23	30	35	23	28	36	33	37	29	23	31	32	25	26	32
MEAN	31.1	27.5	27.5	31.4	29.7	29.3	28.1	33.7	31.1	28.4	27.4	28.8	31.0	31.9	26.7	29.4	32.0
RANGE	11.3	7.0	14.0	12.6	9.6	14.0	6.3	7.9	7.4	14.2	5.7	12.4	8.1	8.8	6.6	10.1	13.1
SUM	155.6	137.3	137.4	157.1	148.3	146.4	140.4	168.4	155.3	142.1	136.9	144.0	155.2	159.3	133.5	147.0	160.2
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

```
GregorianCalendar startTime = new GregorianCalendar();

//  SPC variable control chart type
    int charttype = SPCControlChartData.MEAN_RANGE_CHART;
// Number of samples per sub group
    int numsamplespersubgroup = 5;
// Number of datapoints in the view
    int numdatapointsinview = 17;
// The time increment between adjacent subgroups
    int timeincrementminutes = 15;


public XBarRChart()
{
        // Define and draw chart
        initializeChart();
    }


    public void initializeChart()
    {
        // Initialize the SPCTimeVariableControlChart
        this.initSPCTimeVariableControlChart(charttype,
            numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

        // Change the default horizontal position and width of the chart
        this.setGraphStartPosX ( 0.2); // start here
        this.setGraphStopPosX (0.875);  // end here

        SPCControlChartData chartdata = this.getChartData();
        // Set the strings used in the header section of the table
        chartdata.setTitle ( "Variable Control Chart (X-Bar & R)");
        chartdata.setPartNumber ( "283501");
        chartdata.setChartNumber("17");
        chartdata.setPartName( "Transmission Casing Bolt");
        chartdata.setOperation ( "Threading");
        chartdata.setSpecificationLimits("");
        chartdata.setTheOperator("J. Fenamore");
        chartdata.setMachine("#11");
        chartdata.setGage("#8645");
        chartdata.setUnitOfMeasure ( "0.0001 inch");
        chartdata.setZeroEquals("zero");
        chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));
        chartdata.setNotesMessage ( "Control limits prepared May 10");
        chartdata.setNotesHeader ( "NOTES"); // row header
        this.setHeaderStringsLevel ( SPCControlChartData.HEADER_STRINGS_LEVEL3);
.
.
.

        // Rebuild the chart using the current data and settings
        this.rebuildChartUsingCurrentData();



    }
```

Update the sampled data with your measured values using one of the **addNewSampleRecord** methods.

Method addNewSampleRecord

This method adds a new sample record to the SPC chart. While *both variable control charts* and *attribute control charts* share the same **ChartData addNewSampleRecord** methods, the meaning of the data in the *samples* array varies, depending on the chart type. See the sections below: *Adding New Sample Records for Variable Control Charts*, and *Adding New Sample Records for Attribute Control Charts*.

public void addNewSampleRecord(
  GregorianCalendar *timestamp*,
  DoubleArray *samples*,
  String *notes*
);

Parameters

timestamp
  Time stamp for the current sample record.
samples
  Array of new sample values.
notes
  A string specifying any notes associated with this sample subgroup

controllimits
  Array of control limits, one for each control limits (low, target, and high)

There are many other overloaded versions of **addNewSampleRecord** . Use the one most appropriate to your application.

Adding New Sample Records for Variable Control Charts (Fixed Subgroup Sample Size).

Applies to variable control charts of type: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, INDIVIDUAL_RANGE_CHART, EWMA_CHART, MA_CHART, MAMR_CHART, MAMS_CHART, TABCUSUM_CHART.

In variable control charts, each data value in the *samples* array represents a specific sample in the sample subgroup. In X-Bar R, X-Bar Sigma, and Median-Range charts, where the sample subgroup size is some fraction of the total production level, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. If the production level is sixty items per hour, and the sample size is five items per hour, then the graph would be updated once an hour with five items in the *samples* array.

```
DoubleArray samples = new DoubleArray(5);
//  GregorianCalendar initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
samples.setElement(2, 0.322);      // Third of five samples
samples.setElement(3, 0.021);      // Fourth of five samples
samples.setElement(4, 0.133);      // Fifth of five samples

// Add the new sample subgroup to the chart
this. getChartData().addNewSampleRecord (timestamp, samples);
```

In an Individual-Range chart, which by definition samples 100% of the production level, the *samples* array would only have one value for each update. If the production level is sixty items per hour, with 100% sampling, the graph would be updated once a minute, with a single value in the *samples* array.

Adding New Sample Records to a X-Bar Sigma Chart (Variable Subgroup Sample Size)

Applies to variable control charts of type: MEAN_SIGMA_CHART_VSS

The X-Bar Sigma chart also comes in a version where variable sample sizes are permitted, As in the standard variable control charts, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. The difference is that the length of the *samples* array can change from update to update. It is critically import that the size of the samples array exactly matches the number of samples in the current subgroup

```
// GetCurrentSampleSubgroupSize is a fictional method that gets the
// current number of samples in the  sample subgroup. The value of N
// can vary from sample interval to sample interval. You must have a
// valid sample value for each element.
```

```
N = getCurrentSampleSubgroupSize();

// Size array exactly to a length of N
DoubleArray samples = new DoubleArray(N);
//  GregorianCalendar initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();

// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
samples.setElement(0,0.121);       // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
.
.
.
samples.setElement(N-1, 0.133);   // Last of the samples in the sample subgroup

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord(timestamp, samples);
```

Adding New Sample Records for Attribute Control Charts.

Updating p- and np-charts (Fixed Sample Subgroup Size)

p-chart =	FRACTION_DEFECTIVE_PARTS_CHART or PERCENT_DEFECTIVE_PARTS_CHART
np-chart =	NUMBER_DEFECTIVE_PARTS_CHART

Updating p- and np-charts

In attribute control charts, the meaning of the data in the *samples* array varies, depending on whether the attribute control chart measures the number of defective parts (p-, and np-charts), or the total number of defects (u- and c-charts). The major anomaly is that while the p- and np-charts plot the fraction or number of defective parts, the table portion of the chart can display defect counts for any number of defect categories (i.e. paint scratches, dents, burrs, etc.). It is critical to understand that total number of defects, i.e. the sum of the items in the defect categories for a give sample subgroup, do NOT have to add up to the number of defective parts for the sample subgroup. Every defective part not only can have one or more defects, it can have multiple defects of the same defect category. The total number of defects for a sample subgroup will always be equal to or greater than the number of defective parts. When using p- and np-charts that display defect category counts as part of the table, where N is the *numcategories* parameter in the

initSPCTimeAttributeControlChart or **initSPCBatchAttributeControlChart** initialization call, the first N elements of the *samples* array holds the defect count for each category. The N+1 element of the *samples* array holds the total defective parts count. For example, if you initialized the chart with a *numcategories* parameter to five, signifying that you had five defect categories, you would use a *samples* array sized to six, as in the code below:

```
DoubleArray samples = new DoubleArray(6);
//  GregorianCalendar initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4;    // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5
samples.setElement(5, 4);  // TOTAL number of defective parts in the sample

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord (timestamp, samples);
```

Our example programs obscure this a bit, because we use a special method to simulate defect data for n- and np-charts. The code below is extracted from our TimeAttributeControlCharts.NumberDefectivePartsControlChart example program.

```
DoubleArray samples = this.getChartData().simulateDefectRecord (50 * 0.134,
        SPCControlChartData.NUMBER_DEFECTIVE_PARTS_CHART);
// Add new sample record
this.getChartData().addNewSampleRecord ( timestamp, samples);
```

This particular overload for **getChartData().simulateDefectRecord** knows that since it is a NUMBER_DEFECTIVE_PARTS_CHART chart (np-chart), and since the **ChartData** object was setup with five categories in the **initSPCTimeAttributeControlChart** call, that is should return a **DoubleArray** with (5 + 1 = 6) elements. The first five elements representing simulated defect counts for the five defect categories, and the sixth element the simulated defective parts count. The defect category count data of the *samples* array is only used in the table part of the display; the defect category counts play NO role in the actual SPC chart. The only value plotted in the SPC chart is the last element in the *samples* array, the defective parts count for the sample subgroup.

Updating p-charts (Variable Sample Subgroup Size)

p-chart = FRACTION_DEFECTIVE_PARTS_CHART_VSS
or
PERCENT_DEFECTIVE_PARTS_CHART_VSS

First, you must read the previous section (**Updating p-charts (Fixed Sample Subgroup Size**) and understand it**.** Because in the case of the p-chart variable sample subgroup case, filling out that array is EXACTLY the same as the fixed sample subgroup case. The number of defects in each defect category go into the first N elements (element 0..N-1) of the samples array. The total number of defective parts go into last (element N) of the samples array. Specify the size of the sample subgroup associated with a given update using the **setChartData.SampleSubgroupSize_VSS** method.

```
DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
       SPCControlChartData.NUMBER_DEFECTIVE_PARTS_CHART);

// Randomize the sample subgroup size to some value less than the maximum
// value entered in the call to initSPCTimeAttributeControlChart,
// and set the charts ChartData.SampleSubgroupSize_VSS property with
//  this value immediately prior to the addNewSampleRecord  call.
this.getChartData().setSampleSubgroupSize_VSS(
    numsamplespersubgroup - (int)(25 * ChartSupport.getRandomDouble()));

// Add new sample record
this.getChartData().addNewSampleRecord( timestamp, samples);
```

Updating c- and u-charts (Fixed Sample Subgroup Size)

c-chart = NUMBER_DEFECTS_CHART

u-chart = NUMBER_DEFECTS_PERUNIT_CHART

In c- and u-charts the number of defective parts is of no consequence. The only thing tracked is the number of defects. Therefore, there is no extra array element tacked onto the end of the *samples* array. Each element of the *samples* array represents the total number of defects for a given defect category. If the *numcategories* parameter in the **initSPCTimeAttributeControlChart** or **initSPCBatchAttributeControlChart** is initialized to five, the total number of elements in the *samples* array should be five. For example:

```
DoubleArray samples = new DoubleArray(5);
//  time stamp initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4;    // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord (timestamp, samples);
```

Updating u-charts (Variable Sample Subgroup Size)

u-chart = NUMBER_DEFECTS_PERUNIT_CHART_VSS

First, you must read the previous section (**Updating u-charts Fixed Sample Subgroup Size**) and understand it**.** Because in the case of the u-chart variable sample subgroup case, filling out that array is EXACTLY the same as the fixed sample subgroup case. The number of defects in each defect category go into the first N elements (element 0..N-1) of the samples array. Specify the size of the sample subgroup associated with a given update using the **ChartData.SampleSubgroupSize_VSS** property.

```
DoubleArray samples = new DoubleArray(5);
//  time stamp initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4;    // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

// Randomize the sample subgroup size to some value less than the maximum
// value entered in the call to initSPCTimeAttributeControlChart,
// and set the charts ChartData.SampleSubgroupSize_VSS property with
//  this value immediately prior to the addNewSampleRecord  call.
this.getChartData().setSampleSubgroupSize_VSS(
        numsamplespersubgroup - (int)(25 * ChartSupport.getRandomDouble());

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord(timestamp, samples);
```

While the table portion of the display can display defect data broken down into categories, only the sum of the defects for a given sample subgroup is used in creating the actual SPC chart. Note that the code below, extracted from the TimeAttributeControlCharts.NumberDefectsControlChart example, uses a different **getChartData().simulateDefectRecord** method to simulate the defect data.

```
// Simulate sample record
DoubleArray samples = this.getChartData().simulateDefectRecord (19.85/5);
// Add a sample record
this.getChartData().addNewSampleRecord ( timestamp, samples);
```

Other addNewSampleRecord Methods

Add a new sample record to a time-based SPC chart.

public void addNewSampleRecord(GregorianCalendar,DoubleArray);

Add a new sample record with notes to a time-based SPC chart.

public void addNewSampleRecord(GregorianCalendar,DoubleArray, String);

Add a new sample record to a batch-based SPC chart.

public void addNewSampleRecord(DoubleArray);

Add a new sample record, with notes, to a batch-based SPC chart.

public void addNewSampleRecord(DoubleArray, String);

Add a new sample record to a numeric-based SPC chart.

public void addNewSampleRecord(double,GregorianCalendar,DoubleArray,DoubleArray, String);

Add a new sample record, with notes, to a numeric-based SPC chart .

public void addNewSampleRecord(double,GregorianCalendar,DoubleArray, String);

Add a new sample record, with notes, to a batch-based SPC chart.

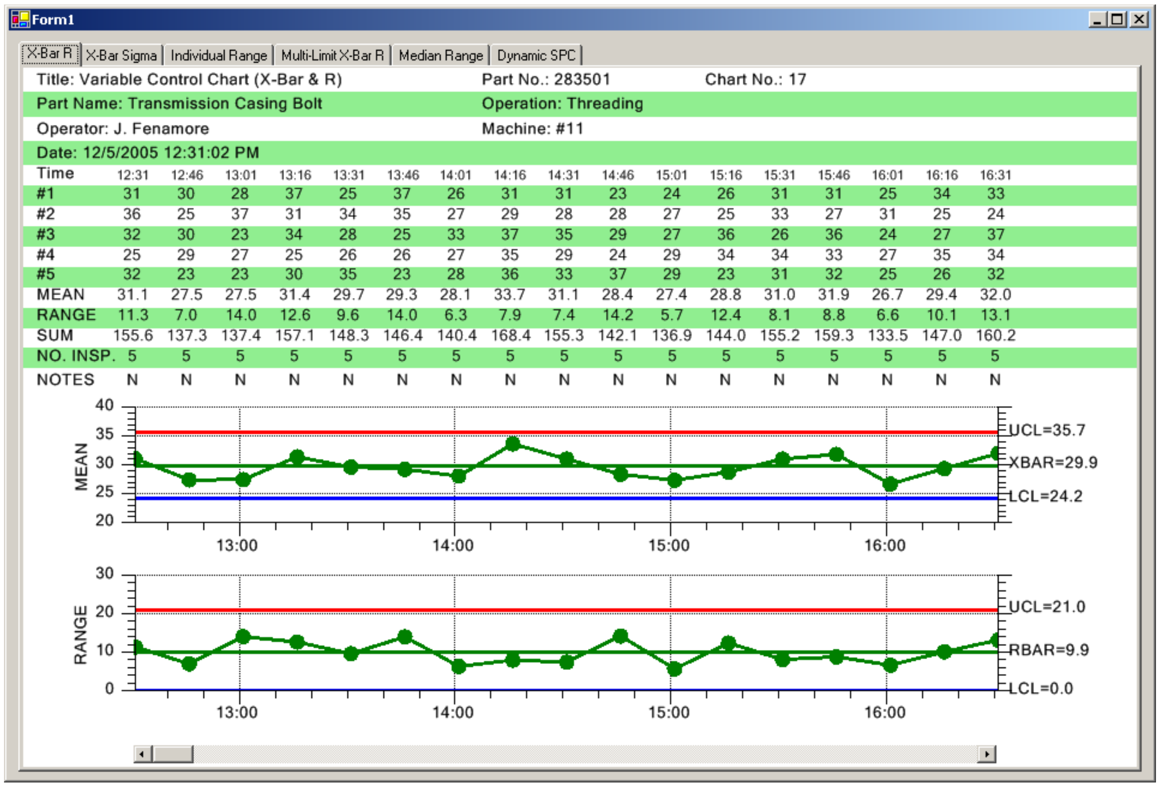
public void addNewSampleRecord(double,DoubleArray);

Add a new sample record, with notes, to a batch-based SPC chart.

public void addNewSampleRecord(double,DoubleArray, String);
In addition to these, there are versions that pass in an additional **DoubleArray** that pass in the current value of variable control limits, if used

If the **addNewSampleRecord** overload does not have an explicit **GregorianCalendar** time stamp parameter, as in the case several of the overloaded methods, the current time as stored in the system clock is used as the time stamp.

The sampled values initialize the chart after the **initSPC**… call, but before the **rebuildChartUsingCurrentData** call. The example below is from the **TimeVariableControlCharts.XBarRChart** example program. The **addNewSampleRecord** routine is called in the **simulateData** method.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 12/5/2005 12:31:02 PM

Time	12:31	12:46	13:01	13:16	13:31	13:46	14:01	14:16	14:31	14:46	15:01	15:16	15:31	15:46	16:01	16:16	16:31
#1	31	30	28	37	25	37	26	31	31	23	24	26	31	31	25	34	33
#2	36	25	37	31	34	35	27	29	28	28	27	25	33	27	31	25	24
#3	32	30	23	34	28	25	33	37	35	29	27	36	26	36	24	27	37
#4	25	29	27	25	26	26	27	35	29	24	29	34	34	33	27	35	34
#5	32	23	23	30	35	23	28	36	33	37	29	23	31	32	25	26	32
MEAN	31.1	27.5	27.5	31.4	29.7	29.3	28.1	33.7	31.1	28.4	27.4	28.8	31.0	31.9	26.7	29.4	32.0
RANGE	11.3	7.0	14.0	12.6	9.6	14.0	6.3	7.9	7.4	14.2	5.7	12.4	8.1	8.8	6.6	10.1	13.1
SUM	155.6	137.3	137.4	157.1	148.3	146.4	140.4	168.4	155.3	142.1	136.9	144.0	155.2	159.3	133.5	147.0	160.2
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

```
public void initializeChart()
{
    // Initialize the SPCTimeVariableControlChart
    this.initSPCTimeVariableControlChart(charttype,
        numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

.
.
.

    // Must have data loaded before any of the Auto.. methods are called
    simulateData();


// Calculate the SPC control limits for both graphs of the current SPC chart
    this.autoCalculateControlLimits();
// Scale the y-axis of the X-Bar chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
// Scale the y-axis of the Range chart to display all data and control limits
    this.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    this.rebuildChartUsingCurrentData();
}



private void simulateData()
{  String notesstring = "";
    for (int i=0; i < 200; i++)
    {
        GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
        // Use the ChartData sample simulator to make an array of sample data

    DoubleArray samples = this.getChartData().simulateMeasurementRecord(30, 10);
        double r = ChartSupport.getRandomDouble();
        if (r < 0.1) // make a note on every tenth item, on average
            notesstring = "Note for sample subgroup #" + Integer.toString(i) + ".
Lathe cutting tool broke. Replaced with new, Aeon cutting tool.";
        else
            notesstring = "";
        // Add the new sample subgroup to the chart
        this.getChartData().addNewSampleRecord(timestamp, samples, notesstring);
        // increment simulated time by timeincrementminutes minutes
        startTime.add(ChartObj.MINUTE, timeincrementminutes);
    }
}
```

Logging SPC Data to a File

The **SPCControlChartData** method contains routines that log SPC data to a file in a CSV (comma separated value) format. The first row of the file is a prefix of data that defines options and the number of columns associated with sample data, calculated data and control limit data. The second row of data are the column heads for each item in the data log. Starting with the third row, SPC data is output, record by record. If the data logging feature is turned on, every call to the **addNewSampleRecord** method will result in the output of that record, and calculated values, to the data log.

A typical datalog file (both the width and length of the data file are truncated) appears below.

```
File prefix:   62,5,3,6
Column Heads:  Time Stamp,Sample #0,Sample #1,Sample #2,…
Record #1:     1/24/2006 12:03:40,22.946081345643,30.6379105980219,…
Record #2:     1/24/2006 12:18:40,23.8902424375481,33.7523682840412,…
Record #3:     1/24/2006 12:33:40,33.1602680593078,28.2172109399537,…
```

The values in the file prefix have the following meaning

63 Data log options = SPCControlChartData.DATALOG_FILE_TIME_STAMP |

SPCControlChartData.DATALOG_FILE_SAMPLED_VALUES |

SPCControlChartData.DATALOG_FILE_CALCULATED_VALUES |

SPCControlChartData.DATALOG_FILE_COLUMN_HEADS |

SPCControlChartData.DATALOG_FILE_NOTES

5 There are five sampled values per record
3 There are three calculated values per record (MEAN, RANGE, SUM for example)
6 There are six control limit values per record (`XBAR,LCL,UCL,RBAR,LCL,UCL`) `for example`

If you want to view a complete datalog file, run the TimeVariableControlCharts example program and after you terminate the program, view the Datalogfile1.text file in the TimeVariableControlCharts\Bin\Debug folder. The following steps, extracted from the TimeVariableControlChart.VariableControlLimitsCharts example program, turn on data logging:

```
int datalogflags = SPCControlChartData.DATALOG_FILE_TIME_STAMP |
    SPCControlChartData.DATALOG_FILE_SAMPLED_VALUES |
    SPCControlChartData.DATALOG_FILE_CALCULATED_VALUES |
    SPCControlChartData.DATALOG_FILE_COLUMN_HEADS |
    SPCControlChartData.DATALOG_FILE_NOTES;


    this.getChartData().dataLogFileOpenForWrite ("DatalogFile1.txt",
datalogflags);
    this.getChartData().setDatalLogEnable(true);
.
.
.
    this.getChartData().addNewSampleRecord (timestamp, samples, notesstring);
```

Every call to the **addNewSampleRecord** method will append a new SPC record to the file specified in the **dataLogFileOpenForWrite** call.

Specify what items are logged to the datalog file using the **setDataLogFlag** method. OR the datalog flags constants together to form the final **DataLogFlags** value.

DATALOG_FILE_ALL	Datalog flag specifying that all available items should be logged to the file.
DATALOG_FILE_BATCH_NUMBER	Datalog flag specifying that the batch number should be logged to the file.
DATALOG_FILE_CALCULATED_VALUES	Datalog flag specifying that the calculated values should be logged to the file.
DATALOG_FILE_COLUMN_HEADS	Datalog flag specifying that the column heads should be logged to the file.
DATALOG_FILE_CONTROL_LIMIT_VALUES	Datalog flag specifying that the control limit values should be logged to the file.
DATALOG_FILE_NOTES	Datalog flag specifying that the notes should be logged to the file.
DATALOG_FILE_SAMPLED_VALUES	Datalog flag specifying that the sampled values should be logged to the file.

DATALOG_FILE_TIME_STAMP	Datalog flag specifying that the time stamp should be logged to the file.

```
this.getChartData().setDataLogFlags(SPCControlChartData.DATALOG_FILE_TIME_STAMP |
    SPCControlChartData.DATALOG_FILE_SAMPLED_VALUES |
    SPCControlChartData.DATALOG_FILE_CALCULATED_VALUES |
    SPCControlChartData.DATALOG_FILE_COLUMN_HEADS |
    SPCControlChartData.DATALOG_FILE_NOTES);
```

It is also possible to read a previously saved datalog file and initialize the **ChartData** object with previously collected data. While the data can be initialized, it is still important that the originating **SPCChartBase** object is initialized properly for the data it is to receive. Use the **getChartData().readAllValuesFromFile** method to read previously saved values. The example below is extracted from the TimeVariableControlChart.VariableControlLimitsCharts example program.

```
this.initSPCTimeVariableControlChart(charttype,
        numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
.
.
.
SPCControlChartData chartdata = this.getChartData();

File filetest = new File("DatalogFile1.txt");
if (filetest.exists())
{
        chartdata.readAllValuesFromFile("DatalogFile1.txt");
}
```

It is important that the *charttype* parameter matches the chart type used to save the original data, and that the *numberofsamplepersubgroup* value matches the number of samples in the original data.

Control Limit Alarms

Class SPCControlLimitRecord

```
ChartObj
        |
        +-- SPCControlLimitRecord
```

The **SPCControlLimitRecord** stores control limit alarm information for the **SPCControlChartData** class. The **SPCControlLimitRecord** class specifies the type of the alarm, the alarm limit value, alarm text messages and alarm hysteresis value. The **SPCControlChartData** classes store the **SPCControlLimitRecord** objects in the **SPCControlChartData.ControlLimitValues** array list

SPCControlLimitRecord constructors

This constructor creates a new instance of a **SPCControlLimitRecord** object, using the specified SPC data object, calculated value object, alarm type, alarm limit value and alarm message.

public **SPCControlLimitRecord**(
  SPCControlChartData *processvar*,
  SPCCalculatedValueRecord *clr*,
  int *parametertype*,
  double *alarmlimitvalue*,
  string *normalmessage*,
  string *alarmmessage*
);

Parameters

processvar
  Specifies the process variable that the alarm is attached to.
clr
  Specifies the calculated value record the alarm is attached to.
parametertype
  Specifies the alarm type: SPC_NOTA_LIMIT, SPC_LOWERTHAN_LIMIT, or SPC_GREATERTHAN_LIMIT.
alarmlimitvalue
  Specifies the alarm limit value.
normalmessage
  Specifies display message when no alarm present.
alarmmessage
  Specifies the alarm message.

Class SPCControlLimitRecord – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCControlChartData.html

a link to the JavaDoc documentation for the class **SPCControlLimitRecord**

The most commonly used **SPCControlLimitRecord** properties are:

Example of trapping SPCControlLimitRecord alarm using an event delegate

The example below specifies an alarm event delegate for the control limit alarms. The example was extracted from the **TimeVariableControlCharts.DynamicXBarRChart** example program.

```
public class DynamicXBarRChart extends SPCTimeVariableControlChart
    implements SPCAlarmEventListener {
    GregorianCalendar startTime = new GregorianCalendar();
    //  SPC variable control chart type
    int charttype = SPCControlChartData.MEAN_RANGE_CHART;
    // Number of samples per sub group
    int numsamplespersubgroup = 5;
    // Number of datapoints in the view
    int numdatapointsinview = 17;
    // The time increment between adjacent subgroups
    int timeincrementminutes = 15;
    });

    public DynamicXBarRChart()
    {
        // Define and draw chart
        initializeChart();
    }


    public void initializeChart()
    {
.
.
.
        this.initSPCTimeVariableControlChart(charttype,
            numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

.
.
.
        // don't generate alarms in initial data simulation
        chartdata.setAlarmStateEventEnable(false);
        simulateData();
.
.
.
        //  generate alarms starting now
        chartdata.setAlarmStateEventEnable( true);
        timer1.start();
    }
```

```
.
.
.
```

```
public void alarmEventChanged(SPCControlChartData sender, SPCControlLimitAlarmArgs
e)
{
    SPCControlLimitRecord alarm = e.getEventAlarm();
    double alarmlimitvalue = alarm.getControlLimitValue();
    String alarmlimitvaluestring = Double.toString(alarmlimitvalue);

    SPCControlChartData spcData = alarm.getSPCProcessVar();
        SPCCalculatedValueRecord spcSource = e.getSPCSource();
        String calculatedvaluestring =
            Double.toString(spcSource.getCalculatedValue());
    String message = alarm.getAlarmMessage();
    GregorianCalendar timestamp = spcData.getTimeStamp();
    String timestampstring = timestamp.toString();

    if (alarm.getAlarmState())
         System.out.println(timestampstring + " " + message + "=" +
        alarmlimitvaluestring + " Current Value" + "=" + calculatedvaluestring);
}
```

Control Limit Alarm Event Handling

Class SPCControlLimitAlarmArgs

ChartObj
        |

+-- SPCControlLimitAlarmArgs

The **SPCControlChartData** class can throw an alarm event based on either the current alarm state, or an alarm transition from one alarm state to another. The **SPCControlLimitAlarmArgs** passes alarm data to the event handler. If you want the alarm event triggered only on the initial transition from the no-alarm state to the alarm state, set the **SPCControlChartData.setAlarmTransitionEventEnable** to true and the **SPCControlChartData.setAlarmStateEventEnable** to false**.** In this case, you will get one event when the process variable goes into alarm, and one when it comes out of alarm. If you want a continuous stream of alarm events, as long as the **SPCControlLimitRecord** object is in alarm, set the **SPCControlChartData.setAlarmTransitionEventEnable** to false and the **SPCControlChartData.setAlarmStateEventEnable** to true. The alarm events will be generated at the same rate as the **SPCControlChartData.addNewSampleRecord ()** method is called.

SPCControlLimitAlarmArgs constructors

You don't really need the constructors since **SPCControlLimitAlarmArgs** objects are created inside the **SPCControlChartData** class when an alarm event needs to be generated.

Class SPCControlLimitAlarmArgs – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCControlLimitAlarmArgs.html

a link to the JavaDoc documentation for the class **SPCControlLimitAlarmArgs**

Example

Setup and enable an alarm transition event handler using the following three steps.

- Add the **implements SPCAlarmEventListener** interface to the chart class.
- Add an implementation of the **SPCAlarmEventListener** method **alarmEventChanged** to the chart class
- Turn on alarm transition alarm events using the ChartData **setAlarmTransitionEventEnable**(true) method.

```
public class DynamicXBarRChart extends SPCTimeVariableControlChart
    implements SPCAlarmEventListener {
```

```
.
.
.

    public void initializeChart()
    {
.
.
.
        this.initSPCTimeVariableControlChart(charttype,
            numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

.
.
.
        // don't generate alarms in initial data simulation
        chartdata.setAlarmTransitionEventEnable(false);

        simulateData();
.
.
.
        //  generate alarms starting now
        chartdata.setAlarmTransitionEventEnable( true);
        timer1.start();
    }
```

The event handler method is the method **alarmEventChanged.**

```
public void alarmEventChanged(SPCControlChartData sender, SPCControlLimitAlarmArgs
e)
{
    SPCControlLimitRecord alarm = e.getEventAlarm();
    double alarmlimitvalue = alarm.getControlLimitValue();
    String alarmlimitvaluestring = Double.toString(alarmlimitvalue);

    SPCControlChartData spcData = alarm.getSPCProcessVar();
.
.
.
}
```

Setup and enable an alarm state event handler in an identical manner:

```
    chartdata.setAlarmStateEventEnable( true);
```

where the handler method is this **alarmEventChanged** method.

```
public void alarmEventChanged(SPCControlChartData sender, SPCControlLimitAlarmArgs
e)
{
    .
    .
    .
}
```

SPCSampledValueRecord

This class encapsulates a sample data value. It includes a description for the item, the current value of the sampled value, and a history of previous values.

An array list of **SPCSampledValueRecord** objects, one for each sample category, is automatically created when the parent **SPCChartBase** object is created. The programmer does not need to instantiate it.

Class SPCSampledValueRecord – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCSampledValueRecord.html

a link to the JavaDoc documentation for the class **SPCSampledValueRecord**

SPCCalculatedValueRecord

This is the record class for a calculated SPC statistic. It holds the calculated value type (mean, median, sum, variance, standard deviation, etc.), value, description and historical data.

Class SPCCalculatedValueRecord – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCCalculatedValueRecord.html

a link to the JavaDoc documentation for the class **SPCCalculatedValueRecord**

SPCProcessCapabilityRecord

This is the record class for calculating and storing SPC process capability statistics. It supports calculating the Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk statistics.

Class SPCProcessCapabilityRecord – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCProcessCapabilityRecord.html

a link to the JavaDoc documentation for the class **SPCProcessCapabilityValueRecord**

SPCGeneralizedTableDisplay

This class manages a list of **ChartText** objects (**NumericLabel**, **StringLabel** and **TimeLabel** objects), that encapsulate each unique table entry in the SPC chart table. This class also manages the spacing between the rows and columns of the table, and the alternating stripe used as a background for the table.

Class SPCGeneralizedTableDisplay – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCGeneralizedTableDisplay.html

a link to the JavaDoc documentation for the class **SPCGeneralizedTableDisplay**

6. SPC Variable Control Charts

SPCTimeVariableControlChart
SPCBatchVariableControlChart

Variable Control Charts are used with sampled quality data that can be assigned a specific numeric value, other than just 0 or 1. This includes, but is not limited to, the measurement of a critical dimension (height, length, width, radius, etc.), the weight a specific component, or the measurement of an important voltage. The variable control charts supported by this software include X-Bar R (Mean and Range), X-Bar Sigma, Median and Range, X-R (Individual Range), MA (Move Average), MAMR (Moving Average / Moving Range), MAMS (Moving Average / Moving Sigma), EWMA (Exponentially Weighted Moving Average) and CUSum charts.

X-Bar R Chart – Also known as the Mean (or Average) and Range Chart

The X-Bar R chart monitors the trend of a critical process variable over time using a statistical sampling method that results in a subgroup of values at each sample interval. The X-Bar part of the chart plots the mean of each sample subgroup and the Range part of the chart monitors the difference between the minimum and maximum value in the subgroup. X-Bar R charts are created using the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

X-Bar Sigma – Also known as the X-Bar S Chart

Very similar to the X-Bar R chart, the X-Bar Sigma chart replaces the Range plot with a Sigma plot based on the standard deviation of the measured values within each subgroup. This is a more accurate way of establishing control limits if the sample size of the subgroup is moderately large (> 10). Though computationally more complicated, the use of a computer makes this a non-issue. The X-Bar Sigma chart comes in fixed sample subgroup size, and variable sample subgroup size, versions. . X-Bar Sigma charts are created using the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

Median Range – Also known as the Median and Range Chart

Very similar to the X-Bar R Chart, Median Range chart replaces the Mean plot with a Median plot representing the median of the measured values within each subgroup. In order to use a Median Range chart the process needs to be well behaved, where the variation in measured variables are (1) known to be distributed normally, (2) are not very often disturbed by assignable causes, and (3) are easily adjusted. . Median Range charts are created using the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

Individual Range Chart – Also known as the X-R Chart

The Individual Range Chart is used when the sample size for a subgroup is 1. This happens frequently when the inspection and collection of data for quality control purposes is automated and 100% of the units manufactured are analyzed. It also happens when the production rate is low and it is inconvenient to have sample sizes other than 1. The X part of the control chart plots the actual sampled value (not a mean or median) for each unit and the R part of the control chart plots a moving range that is calculated using the current value of sampled value minus the previous value. . Individual Range charts are created using the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

EWMA Chart – Exponentially Weighted Moving Average

The EWMA chart is an alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a weighted moving average of previous values to "smooth" the incoming data, minimizing the affect of random noise on the process. It weights the current and most recent values more heavily than older values, allowing the control line to react faster than a simple MA (Moving Average) plot to changes in the process. Like the Shewhart charts, if the EWMA value exceeds the calculated control limits, the process is considered out of control. While it is usually used where the process uses 100% inspection and the sample subgroup size is 1 (same is the X-R chart), it can also be used when sample subgroup sizes are greater than one. EWMA charts are created using the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

MA Chart – Moving Average

The MA chart is another alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a moving average, where the previous (N-1) sample values of the process variable are averaged together along with the current value to produce the current chart value. This helps to "smooth" the incoming data, minimizing the affect of random noise on the process. Unlike the EWMA chart, the MA chart weights the current and previous (N-1) values equally in the average. While the MA chart can often detect small changes in the process mean faster than the Shewhart chart types, it is generally less effective than either the EWMA chart, or the CuSum chart. MA charts are created using the **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

MAMR Chart – Moving Average/Moving Range

The MAMR chart combines our Moving Average chart with a Moving Range chart. The Moving Average chart is primary (topmost) chart, and the Moving Range chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Range, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving range statistics.

MAMS Chart – Moving Average / Moving Sigma

The MAMS chart combines our Moving Average chart with a Moving Sigma chart. The Moving Average chart is primary (topmost) chart, and the Moving Sigma chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Sigma, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving sigma statistics.

CuSum Chart – Tabular, one-sided, upper and lower cumulative sum

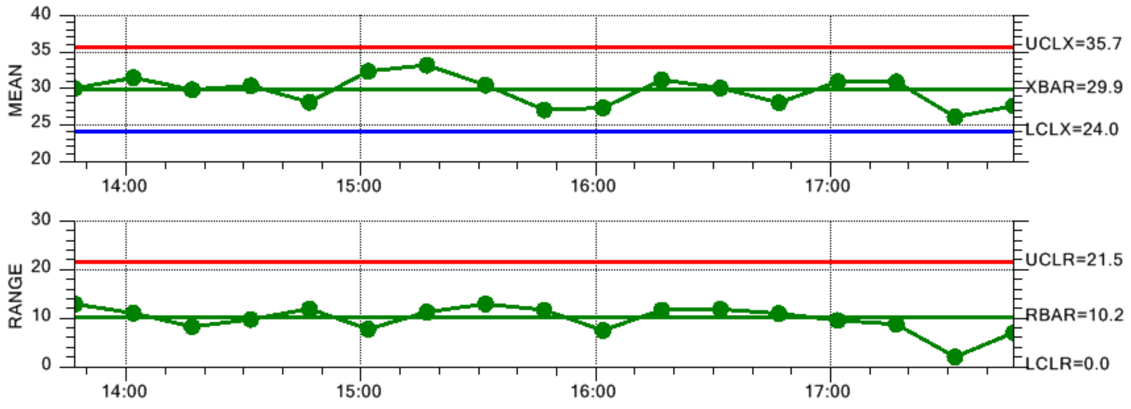
The CuSum chart is a specialized control chart, which like the EWMA and MA charts, is considered to be more efficient that the Shewhart charts at detecting small shifts in the process mean, particularly if the mean shift is less than 2 sigma. There are several types of CuSum charts, but the easiest to use and the most accurate is considered the tabular CuSum chart and that is the one implemented in this software. The tabular cusum works by accumulating deviations that are above the process mean in one statistic (C+) and accumulating deviations below the process mean in a second statistic (C-). If either statistic (C+ or C-) falls outside of the calculated limits, the process is considered out of control.

Time-Based and Batch-Based SPC Charts

The **QCSPCChart** software further categorizes *Variable Control* as either time- or batch- based. Time-based SPC charts are used when data is collected using a subgroup interval corresponding to a specific time interval. Batch-based SPC charts are used when the data subgroup interval is a sequential batch number that does not correspond to a uniform time interval. The major difference in these two types of SPC charts is the display of the x-axis. Variable control charts that sample using a uniform time interval will generally use a time-based x-axis, with time/date axis labels. Variable control charts that sample based on batches will generally use a numeric-based x-axis, with numeric axis labels.

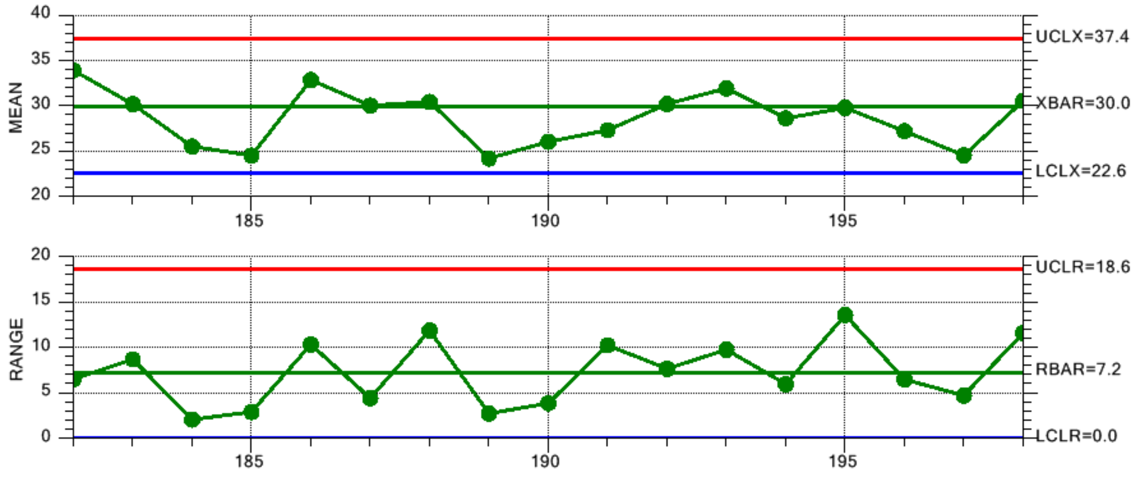
Note: Starting with Revision 2.0, batch control charts can label the x-axis using one of three options: numeric labeling (the original and default mode), time stamp labeling, and user defined string labeling. Since this affects batch control charts, time stamps to not have to be equally spaced, or even sequential

Time-Based Variable Control Chart



Note the time-based x-axis for both charts.

Batch-Based Variable Control Chart



Note the numeric based x-axis for both graphs

Creating a Variable Control Chart

First, select whether you want to use a time-based variable control chart (use **SPCTimeVariableControlChart**) or a batch-based variable control chart (use **SPCBatchVariableControlChart**). Use that class as the base class for your chart. Since the two classes are so similar and share 95% of all properties in common, only the **SPCTimeVariableControlChart** is discussed in detail, with the differences between the two classes discussed at the end of the chapter. The example below is extracted from the TimeVariableControlCharts.XBarRChart example program.

```
public class XBarRChart extends SPCTimeVariableControlChart  {

       GregorianCalendar startTime = new GregorianCalendar();

       //  SPC variable control chart type
              int charttype = SPCControlChartData.MEAN_RANGE_CHART;
       // Number of samples per sub group
              int numsamplespersubgroup = 5;
       // Number of datapoints in the view
              int numdatapointsinview = 17;
       // The time increment between adjacent subgroups
              int timeincrementminutes = 15;


       public XBarRChart()
       {

              // Define and draw chart
              initializeChart();
       }


       public void initializeChart()
       {
              // Initialize the SPCTimeVariableControlChart
              this.initSPCTimeVariableControlChart(charttype,
                     numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

.
.
.

              // Rebuild the chart using the current data and settings
              this.rebuildChartUsingCurrentData();


       }
```

SPCTimeVariableControlChart Members

Class SPCTimeVariableControlChart – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCTimeVariableControlChart.html

a link to the JavaDoc documentation for the class **SPCTimeVariableControlChart**

The control chart type (X-Bar R, Median Range, X-Bar Sigma, Individual Range, EWMA, MA) establishes the variable control charts **initSPCTimeVariableControlChart** initialization routine. Note that the X-Bar Sigma chart, with a variable subgroup sample size, is initialized using **initSPCTimeVariableControlChart** with a *charttype* value of MEAN_SIGMA_CHART_VSS. X-Bar Sigma charts with sub groups that use a variable sample size must be updated properly. See the section “Adding New Sample Records to a

X-Bar Sigma Chart (Variable Subgroup Sample Size)" in the "SPC Control Data and Alarm Classes" chapter.

SPCTimeVariableControlChart.initSPCTimeVariableControlChart Method

This initialization method initializes the most important values in the creation of a SPC chart.

```
public void initSPCTimeVariableControlChart(
   int charttype,
   int numsamplespersubgroup,
   int numdatapointsinview,
   int timeincrementminutes
);
```

Parameters

charttype

The SPC chart type parameter. Use one of the SPCControlChartData SPC chart types: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART and MAMS_CHART chart types

numsamplespersubgroup

Specifies the number of samples that make up a sample subgroup.

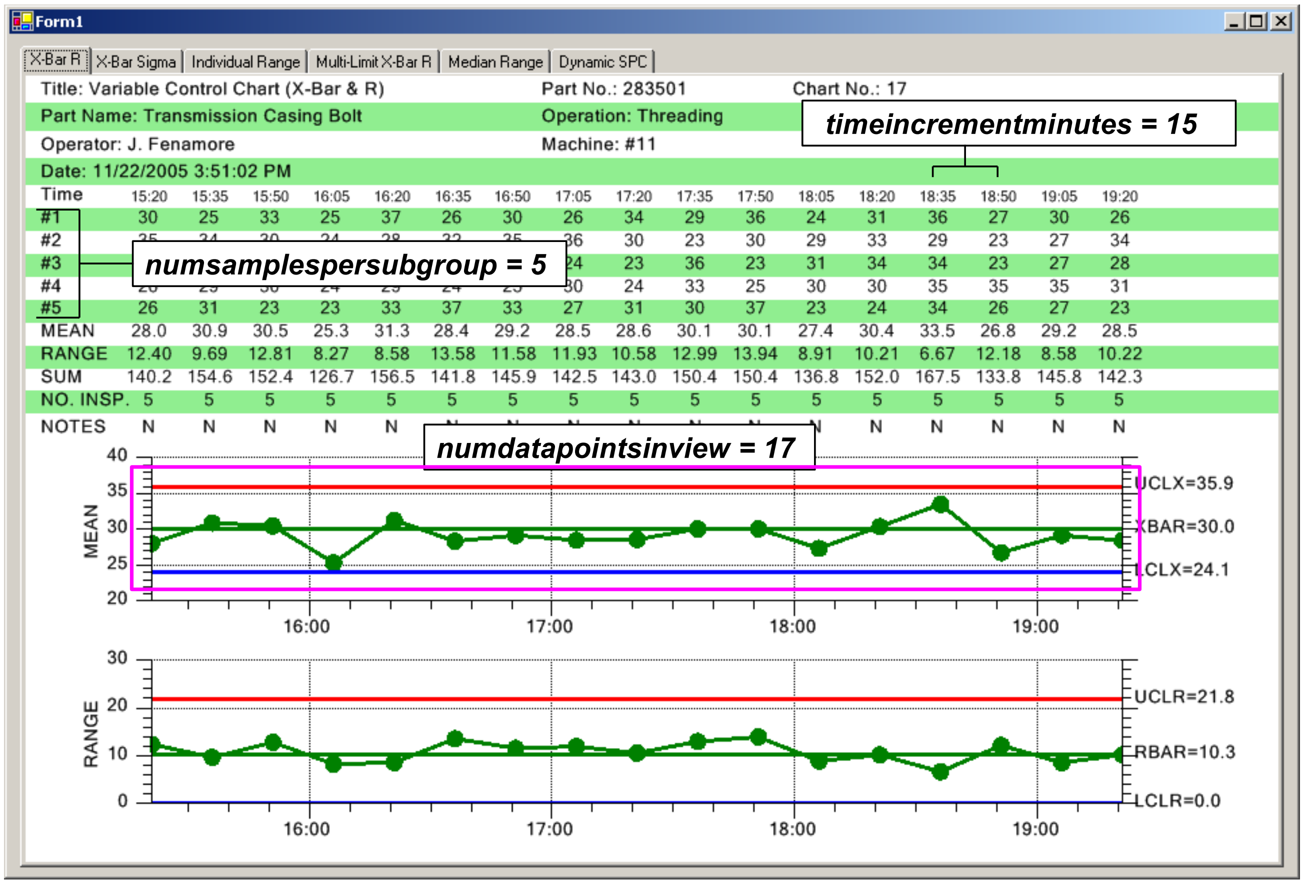
numdatapointsinview

Specifies the number of sample subgroups displayed in the graph at one time.

timeincrementminutes

Specifies the normal time increment between adjacent subgroup samples.

The image below further clarifies how these parameters affect the variable control chart.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 11/22/2005 3:51:02 PM

Time	15:20	15:35	15:50	16:05	16:20	16:35	16:50	17:05	17:20	17:35	17:50	18:05	18:20	18:35	18:50	19:05	19:20
#1	30	25	33	25	37	26	30	26	34	29	36	24	31	36	27	30	26
#2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	36	30	23	30	29	33	29	23	27	34
#3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	24	23	36	23	31	34	34	23	27	28
#4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	30	24	33	25	30	30	35	35	35	31
#5	26	31	23	23	33	37	33	27	31	30	37	23	24	34	26	27	23
MEAN	28.0	30.9	30.5	25.3	31.3	28.4	29.2	28.5	28.6	30.1	30.1	27.4	30.4	33.5	26.8	29.2	28.5
RANGE	12.40	9.69	12.81	8.27	8.58	13.58	11.58	11.93	10.58	12.99	13.94	8.91	10.21	6.67	12.18	8.58	10.22
SUM	140.2	154.6	152.4	126.7	156.5	141.8	145.9	142.5	143.0	150.4	150.4	136.8	152.0	167.5	133.8	145.8	142.3
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	N	N	N	N	N	N

Once the init routine is called, the chart can be further customized using properties inherited from **SPCChartChart**, described below.

Class SPCChartBase – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCChartBase.html

a link to the JavaDoc documentation for the class **SPCChartBase**

Adding New Sample Records for Variable Control Charts.

In variable control charts, each data value in the *samples* array represents a specific sample in the sample subgroup. In X-Bar R, X-Bar Sigma, and Median-Range charts, where the sample subgroup size is some fraction of the total production level, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. If the production level is sixty items per hour, and the sample size is five items per hour, then the graph would be updated once an hour with five items in the *samples* array.

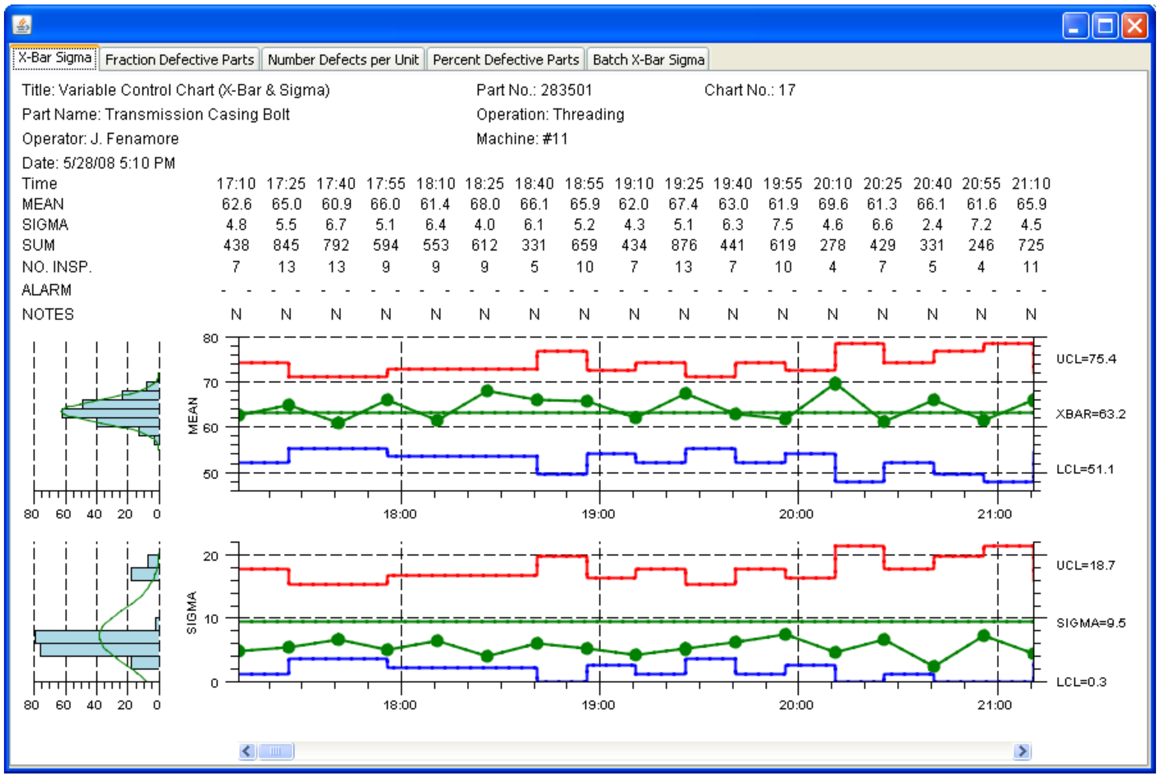
```
DoubleArray samples = new DoubleArray(5);
//  time stamp initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0,0.121);       // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
samples.setElement(2, 0.322);      // Third of five samples
samples.setElement(3, 0.021);      // Fourth of five samples
samples.setElement(4, 0.133);      // Fifth of five samples

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord (timestamp, samples);
```

In an Individual-Range chart, which by definition samples 100% of the production level, the *samples* array would only have one value for each update. If the production level is sixty items per hour, with 100% sampling, the graph would be updated once a minute, with a single value in the *samples* array.

Updating MEAN_SIGMA_CHART_VSS with a variable number of samples per subgroup

The X-Bar Sigma chart also comes in a version where variable sample sizes are permitted, As in the standard variable control charts, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. The difference is that the length of the *samples* array can change from update to update. It is critically import that the size of the samples array exactly matches the number of samples in the current subgroup

X-Bar Sigma Chart with variable sample size



Title: Variable Control Chart (X-Bar & Sigma) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 5/28/08 5:10 PM

Time	17:10	17:25	17:40	17:55	18:10	18:25	18:40	18:55	19:10	19:25	19:40	19:55	20:10	20:25	20:40	20:55	21:10
MEAN	62.6	65.0	60.9	66.0	61.4	68.0	66.1	65.9	62.0	67.4	63.0	61.9	69.6	61.3	66.1	61.6	65.9
SIGMA	4.8	5.5	6.7	5.1	6.4	4.0	6.1	5.2	4.3	5.1	6.3	7.5	4.6	6.6	2.4	7.2	4.5
SUM	438	845	792	594	553	612	331	659	434	876	441	619	278	429	331	246	725
NO. INSP.	7	13	13	9	9	9	5	10	7	13	7	10	4	7	5	4	11
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. You can read the sample sizes along the NO.INSP row in the data table above the chart. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are available. The X-Bar Sigma chart is the only variable control chart that can be used with a variable sample size.

```
// getCurrentSampleSubgroupSize is a fictional method that gets the
// current number of samples in the  sample subgroup. The value of N
// can vary from sample interval to sample interval. You must have a
// valid sample value for each element.

N = getCurrentSampleSubgroupSize();

// Size array exactly to a length of N
DoubleArray samples = new DoubleArray(N);
//  time stamp initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();

// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
```

```
samples.setElement(0,0.121);       // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
.
.
.
samples.setElement(N-1, 0.133);   // Last of the samples in the sample subgroup

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord(timestamp, samples);
```

Measured Data and Calculated Value Tables

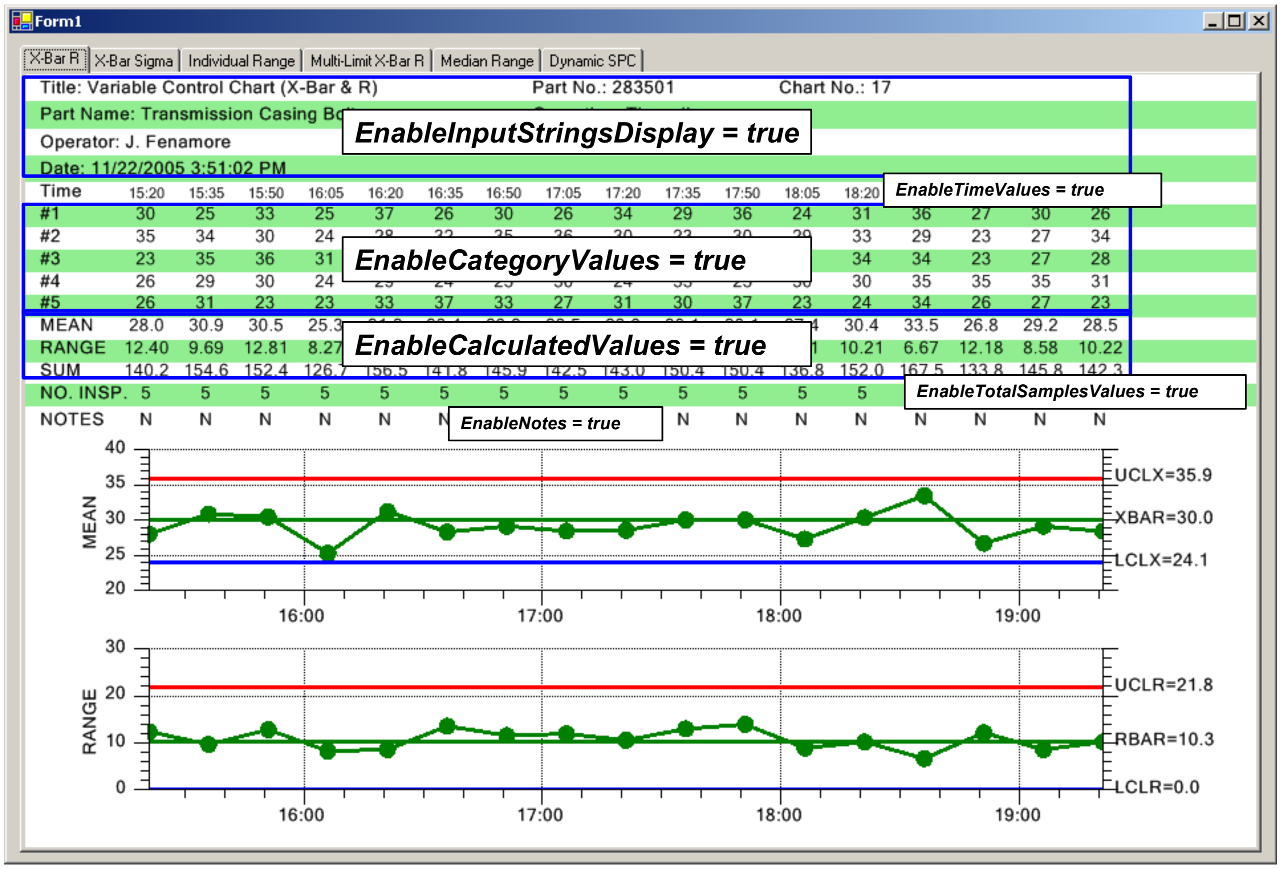
Standard worksheets used to gather and plot SPC data consist of three main parts.

- The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.
- The second part is the measurement data recording and calculation section, organized as a table where the sample data and calculated values are recorded in a neat, readable fashion.
- The third part plots the calculated SPC values for the sample group variables as a SPC chart.

The chart includes options that enable the programmer to customize and automatically include header information along with a table of the measurement and calculated data, in the SPC chart.

The following methods enable sections of the chart header and table:
:

setEnableInputStringsDisplay
setEnableCategoryValues
setEnableCalculatedValues
setEnableTotalSamplesValues
setEnableNotes
setEnableTotalSamplesValues
setEnableTimeValues



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bo
Operator: J. Fenamore
Date: 11/22/2005 3:51:02 PM

Time	15:20	15:35	15:50	16:05	16:20	16:35	16:50	17:05	17:20	17:35	17:50	18:05	18:20	[illegible]	[illegible]	[illegible]	[illegible]
#1	30	25	33	25	37	26	30	26	34	29	36	24	31	36	27	30	26
#2	35	34	30	24	28	32	35	26	30	22	30	29	33	29	23	27	34
#3	23	35	36	31	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	34	34	23	27	28
#4	26	29	30	24	25	24	25	30	24	33	25	30	30	35	35	35	31
#5	26	31	23	23	33	37	33	27	31	30	37	23	24	34	26	27	23
MEAN	28.0	30.9	30.5	25.3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	27.4	30.4	33.5	26.8	29.2	28.5
RANGE	12.40	9.69	12.81	8.27	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	10.21	6.67	12.18	8.58	10.22
SUM	140.2	154.6	152.4	126.7	156.5	141.8	145.9	142.5	143.0	150.4	150.4	136.8	152.0	167.5	133.8	145.8	142.3
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	[illegible]	[illegible]	[illegible]	[illegible]
NOTES	N	N	N	N	N	N	[illegible]	[illegible]	[illegible]	N	N	N	N	N	N	N	N

In the program the code looks like the following code extracted from the TimeVariableControlCharts.XBarRChart example program

```
// Initialize the SPCTimeVariableControlChart
this.initSPCTimeVariableControlChart(charttype,
           numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

// Change the default horizontal position and width of the chart
this.setGraphStartPosX ( 0.2); // start here
this.setGraphStopPosX (0.875);  // end here

SPCControlChartData chartdata = this.getChartData();
// Set the strings used in the header section of the table
chartdata.setTitle ( "Variable Control Chart (X-Bar & R)");
chartdata.setPartNumber ( "283501");
chartdata.setChartNumber("17");
chartdata.setPartName( "Transmission Casing Bolt");
chartdata.setOperation ( "Threading");
chartdata.setSpecificationLimits("");
chartdata.setTheOperator("J. Fenamore");
chartdata.setMachine("#11");
chartdata.setGage("#8645");
chartdata.setUnitOfMeasure ( "0.0001 inch");
chartdata.setZeroEquals("zero");
chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));
chartdata.setNotesMessage ( "Control limits prepared May 10");
chartdata.setNotesHeader ( "NOTES"); // row header
this.setHeaderStringsLevel ( SPCControlChartData.HEADER_STRINGS_LEVEL3);


SPCChartObjects primaryChart = this.getPrimaryChart();
SPCChartObjects secondaryChart = this.getSecondaryChart();

// Set initial scale of the y-axis of the mean chart
// If you are calling AutoScalePrimaryChartYRange this isn't really needed
primaryChart.setMinY ( 0);
primaryChart.setMaxY ( 40);

// Set initial scale of the y-axis of the range chart
// If you are calling AutoScaleSecondaryChartYRange this isn't really needed
secondaryChart.setMinY ( 0);
secondaryChart.setMaxY ( 40);


// Display the Sampled value rows of the table
this.setEnableInputStringsDisplay( true);
// Display the Sampled value rows of the table
this.setEnableCategoryValues( true);
// Display the Calculated value rows of the table
this.setEnableCalculatedValues( true);
// Display the total samples per subgroup value row
this.setEnableTotalSamplesValues( false);
// Display the Notes row of the table
this.setEnableNotes( true);
// Display the time stamp row of the table
this.setEnableTimeValues ( true);
```

Process Capability Ratios and Process Performance Indices

The data table also displays any process capability statistics that you want to see. The software supports the calculation and display of the Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppu, Ppl, and Ppk process capability statistics.

In order to display process capability statistics you must first specify the process specification limits that you want the calculations based on. These are not the high and low SPC control limits calculate by this software; rather they externally calculated limits based on the acceptable tolerances allowed for the process under measure. Set the lower specification limit (LSL) and upper specification limit (USL) using the **ChartData.ProcessCapabilityLSLValue** and **ChartData.ProcessCapabilityUSLValue** properties of the chart. The code below is from the TimeVariableControlCharts.XBarRChart example.

```
chartdata.setProcessCapabilityLSLValue(27);
chartdata.setProcessCapabilityUSLValue(35);
```

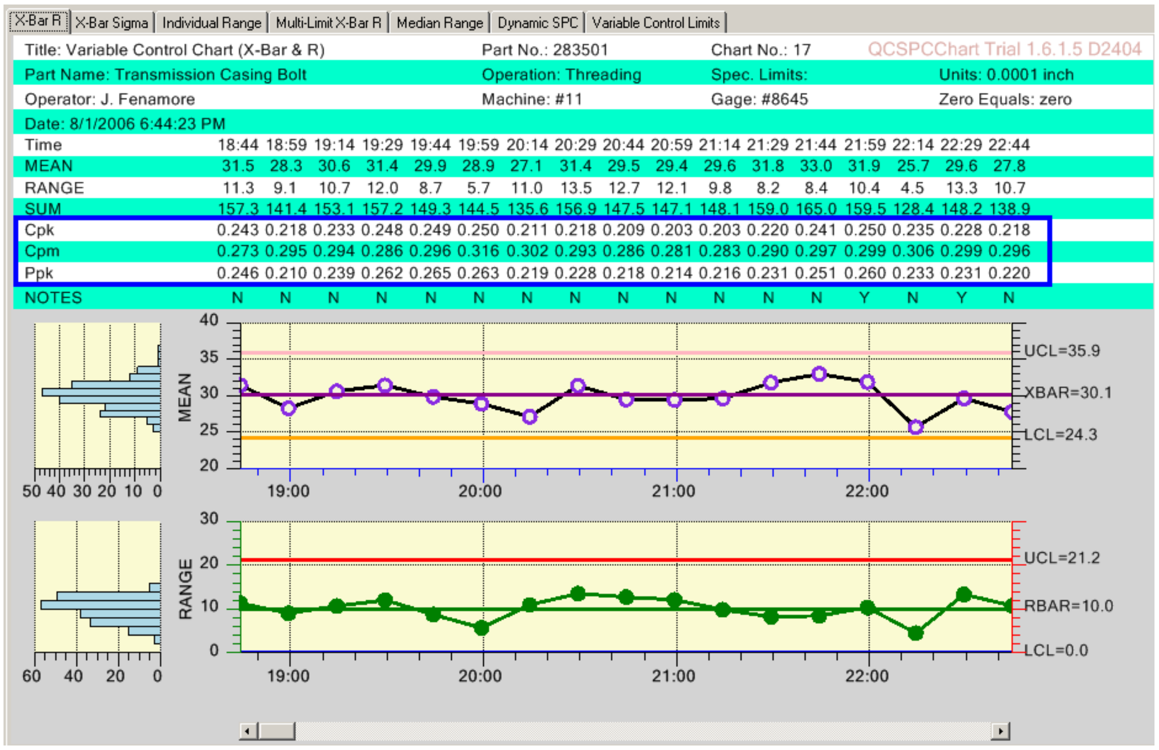
Use the **ChartData.addProcessCapabilityValue** method to specify exactly which process capability statistics you want to see in the table. Use one of the **SPCProcessCapabilityRecord** constants below to specify the statistics that you want displayed.

```
      SPC_CP_CALC        Constant value Cp calculation
      SPC_CPL_CALC       Constant value Cpl calculation.
      SPC_CPU_CALC       Constant value Cpu calculation.
      SPC_CPK_CALC       Constant value Cpk calculation.
      SPC_CPM_CALC       Constant value Cpm calculation.
      SPC_PP_CALC        Constant value Pp calculation.
      SPC_PPL_CALC       Constant value Ppl calculation.
      SPC_PPU_CALC       Constant value Ppu calculation.
      SPC_PPK_CALC       Constant value PPK calculation.
```

The code below is from the TimeVariableControlCharts.XBarRChart example.

```
chartdata.addProcessCapabilityValue(SPCProcessCapabilityRecord.SPC_CPK_CALC);
chartdata.addProcessCapabilityValue(SPCProcessCapabilityRecord.SPC_CPM_CALC);
chartdata.addProcessCapabilityValue(SPCProcessCapabilityRecord.SPC_PPK_CALC);
```

This selection will add three rows to the data table, one row each for the Cpk, Cpm and Ppk process capability statistics. Once these steps are carried out, the calculation and display of the statistics is automatic.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch

Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero

Date: 8/1/2006 6:44:23 PM

Time	18:44	18:59	19:14	19:29	19:44	19:59	20:14	20:29	20:44	20:59	21:14	21:29	21:44	21:59	22:14	22:29	22:44
MEAN	31.5	28.3	30.6	31.4	29.9	28.9	27.1	31.4	29.5	29.4	29.6	31.8	33.0	31.9	25.7	29.6	27.8
RANGE	11.3	9.1	10.7	12.0	8.7	5.7	11.0	13.5	12.7	12.1	9.8	8.2	8.4	10.4	4.5	13.3	10.7
SUM	157.3	141.4	153.1	157.2	149.3	144.5	135.6	156.9	147.5	147.1	148.1	159.0	165.0	159.5	128.4	148.2	138.9
Cpk	0.243	0.218	0.233	0.248	0.249	0.250	0.211	0.218	0.209	0.203	0.203	0.220	0.241	0.250	0.235	0.228	0.218
Cpm	0.273	0.295	0.294	0.286	0.296	0.316	0.302	0.293	0.286	0.281	0.283	0.290	0.297	0.299	0.306	0.299	0.296
Ppk	0.246	0.210	0.239	0.262	0.265	0.263	0.219	0.228	0.218	0.214	0.216	0.231	0.251	0.260	0.233	0.231	0.220
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	Y	N

Formulas Used in Calculating the Process Capability Ratios

The formulas used in calculating the process capability statistics vary. We use the formulas found in the textbook. “Introduction to Statistical Quality Control” by Douglas C. Montgomery, John Wiley and Sons, Inc. 2001.

SPC Control Chart Nomenclature

USL = Upper Specification Limit

LSL = Lower Specification Limit

Tau = Midpoint between USL and LSL = ½ * (LSL + USL)

$\overline{\overline{X}}$ = XDoubleBar - Mean of sample subgroup means (also called the grand average)

$\overline{R}$ = RBar – Mean of sample subgroup ranges

S = Sigma – sample standard deviation – all samples from all subgroups are used to calculate the standard deviation S.

$\overline{S}$ = SigmaBar – Average of sample subgroup sigma's. Each sample subgroup has a calculated standard deviation and the SigmaBar value is the mean of those subgroup standard deviations.

d2 = a constant tabulated in every SPC textbook for various sample sizes.

By convention, the quantity RBar/d2 is used to estimate the process sigma for the Cp, Cpl and Cpu calculations

MINIMUM – a function that returns the lesser of two arguments

SQRT – a function returning the square root of the argument.

Process Capability Ratios (Cp, Cpl, Cpu, Cpk and Cpm)

Cp = (USL – LSL) / (6 * RBar/d2)

Cpl = (XDoubleBar – LSL) / (3 * RBar/d2)

Cpu = (USL - XDoubleBar) / (3 * RBar/d2)

Cpk = MINIMUM (Cpl, Cpu)

Cpm = Cp / (SQRT(1 + V^2)

where

V = (XDoubleBar – Tau) / S

Process Performance Indices (Pp, Ppl, Ppu, Ppk)

Pp = (USL – LSL) / (6 * S)

Ppl	=	(XDoubleBar – LSL) / (3 * S)
Ppu	=	(USL - XDoubleBar) / (3 *S)
Ppk	=	MINIMUM (Ppl, Ppu)

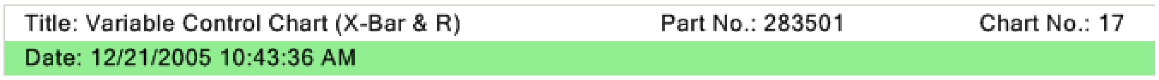
The major difference between the Process Capability Ratios (Cp, Cpl, Cpu, Cpk) and the Process Performance Indices (Pp, Ppl, Ppu, Ppk) is the estimate used for the process sigma. The Process Capability Ratios use the estimate (RBar/d2) and the Process Performance Indices uses the sample standard deviation S. If the process is in control, then Cp vs Pp and Cpk vs Ppk should returns approximately the same values, since both (RBar/d2) and the sample sigma S will be good estimates of the overall process sigma. If the process is NOT in control, then ANSI (American National Standards Institute) recommends that the Process Performance Indices (Pp, Ppl, Ppu, Ppk) be used.

Table Strings

The input header strings display has four sub-levels that display increasing levels of information. The input header strings display level is set using the charts HeaderStringsLevel property. Strings that can be displayed are: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString. The four levels and the information displayed is listed below:

HEADER_STRINGS_LEVEL0	Display no header information
HEADER_STRINGS_LEVEL1	Display minimal header information: Title, PartNumber, ChartNumber, DateString
HEADER_STRINGS_LEVEL2	Display most header strings: Title, PartNumber, ChartNumber, PartName, Operation, Operator, Machine, DateString
HEADER_STRINGS_LEVEL3	Display all header strings: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString

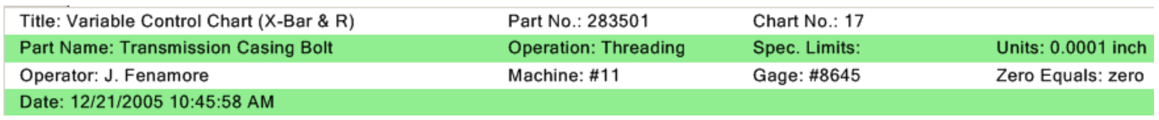
The example program TimeVariableControlCharts.XBarRChart demonstrates the use of the **HeaderStringsLevel** property. The example below displays a minimum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL1).



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17
Date: 12/21/2005 10:43:36 AM		

```
SPCControlChartData chartdata = this.getChartData();
// Set the strings used in the header section of the table
chartdata.setTitle ( "Variable Control Chart (X-Bar & R)");
chartdata.setPartNumber ( "283501");
chartdata.setChartNumber("17");
chartdata.setPartName( "Transmission Casing Bolt");
chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));
this.setHeaderStringsLevel ( SPCControlChartData.HEADER_STRINGS_LEVEL1);
```

The example below displays a maximum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL3).



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation: Threading	Spec. Limits:	Units: 0.0001 inch
Operator: J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 12/21/2005 10:45:58 AM			

```
SPCControlChartData chartdata = this.getChartData();
// Set the strings used in the header section of the table
chartdata.setTitle ( "Variable Control Chart (X-Bar & R)");
chartdata.setPartNumber ( "283501");
chartdata.setChartNumber("17");
chartdata.setPartName( "Transmission Casing Bolt");
chartdata.setOperation ( "Threading");
chartdata.setSpecificationLimits("");
chartdata.setTheOperator("J. Fenamore");
chartdata.setMachine("#11");
chartdata.setGage("#8645");
chartdata.setUnitOfMeasure ( "0.0001 inch");
chartdata.setZeroEquals("zero");
chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));
chartdata.setNotesMessage ( "Control limits prepared May 10");
chartdata.setNotesHeader ( "NOTES"); // row header
this.setHeaderStringsLevel ( SPCControlChartData.HEADER_STRINGS_LEVEL3);
```

The identifying string displayed in front of the input header string can be any string that you want, including non-English language strings. For example, if you want the input header string for the Title to represent a project name:

Project Name: Project XKYZ for PerQuet

Set the properties:

```
this.getChartData().setTitle("Project XKYZ for PerQuet");
this.getChartData().setTitleHeader(Project Name:");
```

Change other header strings using the **ChartData** properties listed below.

- TitleHeader
- PartNumberHeader
- ChartNumberHeader
- PartNameHeader
- OperationHeader
- OperatorHeader
- MachineHeader
- DateHeader
- SpecificationLimitsHeader
- GageHeader
- UnitOfMeasureHeader
- ZeroEqualsHeader
- NotesHeader

Even though the input header string properties have names like Title, PartNumber, ChartNumber, etc., those names are arbitrary. They are really just placeholders for the strings that are placed at the respective position in the table. You can display any combination of strings that you want, rather than the ones we have selected by default, based on commonly used standardized SPC Control Charts.

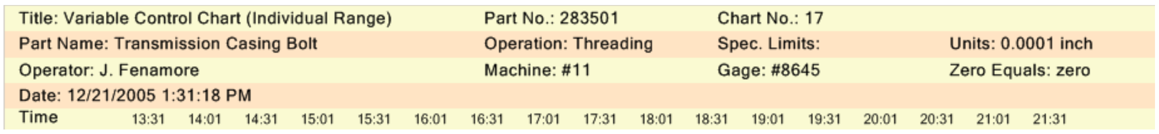
Table Background Colors

The **ChartTable** property of the chart has some properties that can further customize the chart. The default table background uses the accounting style green-bar striped background. You can change this using the **ChartTable.TableBackgroundMode** property**.** Set the value to one of the TableBackgroundMode constants in the class **SPCGeneralizedTableDisplay**:

TABLE_NO_COLOR_BACKGROUND	Constant specifies that the table does not use a background color.
TABLE_SINGLE_COLOR_BACKGROUND	Constant specifies that the table uses a single color for the background (backgroundColor1)
TABLE_STRIPED_COLOR_BACKGROUND	Constant specifies that the table uses horizontal stripes of color for the background (backgroundColor1 and backgroundColor2)

TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL
Constant specifies that the table uses a grid background, with backgroundColor1 the overall background color and backgroundColor2 the color of the grid lines.

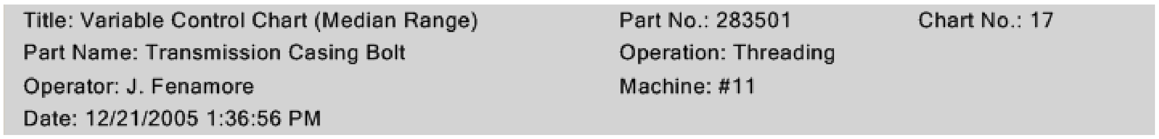
Extracted from the TimeVariableControlCharts.IndividualRangeChart example program



Title: Variable Control Chart (Individual Range)								Part No.: 283501				Chart No.: 17				
Part Name: Transmission Casing Bolt								Operation: Threading				Spec. Limits:				Units: 0.0001 inch
Operator: J. Fenamore								Machine: #11				Gage: #8645				Zero Equals: zero
Date: 12/21/2005 1:31:18 PM																
Time	13:31	14:01	14:31	15:01	15:31	16:01	16:31	17:01	17:31	18:01	18:31	19:01	19:31	20:01	20:31	21:01	21:31

```
SPCGeneralizedTableDisplay chartTable = this.getChartTable();


chartTable.setTableBackgroundMode (
        SPCGeneralizedTableDisplay.TABLE_STRIPED_COLOR_BACKGROUND);
chartTable.setBackgroundColor1 ( ChartColors.BEIGE);
chartTable.setBackgroundColor2 ( ChartColors.LIGHTGOLDENRODYELLOW);
```

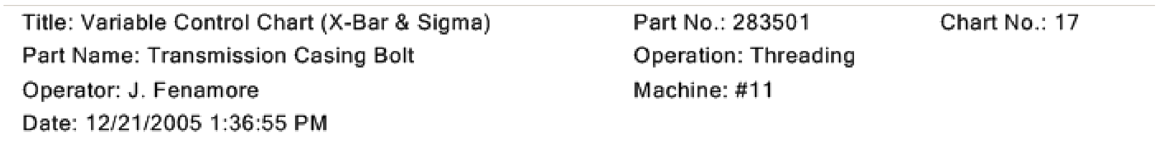
Extracted from the TimeVariableControlCharts.MedianRangeChart example program



Title: Variable Control Chart (Median Range)	Part No.: 283501	Chart No.: 17
Part Name: Transmission Casing Bolt	Operation: Threading	
Operator: J. Fenamore	Machine: #11	
Date: 12/21/2005 1:36:56 PM		

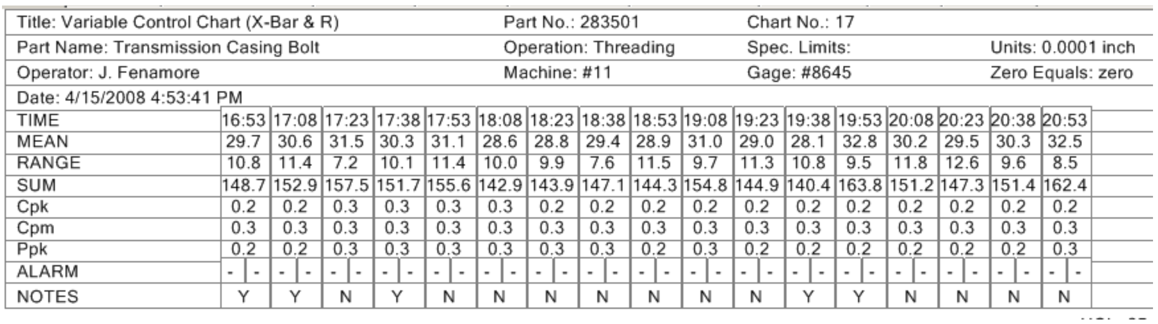
```
this.getChartTable().setTableBackgroundMode (

                SPCGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND);

this.getChartTable().setBackgroundColor1 ( ChartColors.LIGHTGREY);
```

Extracted from the TimeVariableControlCharts.XBarSigma example program



Title: Variable Control Chart (X-Bar & Sigma)	Part No.: 283501	Chart No.: 17
Part Name: Transmission Casing Bolt	Operation: Threading	
Operator: J. Fenamore	Machine: #11	
Date: 12/21/2005 1:36:55 PM		

```
this.getChartTable().setTableBackgroundMode

                (SPCGeneralizedTableDisplay.TABLE_NO_COLOR_BACKGROUND);
```



Title: Variable Control Chart (X-Bar & R) | Part No.: 283501 | Chart No.: 17

Part Name: Transmission Casing Bolt | Operation: Threading | Spec. Limits: | Units: 0.0001 inch

Operator: J. Fenamore | Machine: #11 | Gage: #8645 | Zero Equals: zero

Date: 4/15/2008 4:53:41 PM

TIME	16:53	17:08	17:23	17:38	17:53	18:08	18:23	18:38	18:53	19:08	19:23	19:38	19:53	20:08	20:23	20:38	20:53
MEAN	29.7	30.6	31.5	30.3	31.1	28.6	28.8	29.4	28.9	31.0	29.0	28.1	32.8	30.2	29.5	30.3	32.5
RANGE	10.8	11.4	7.2	10.1	11.4	10.0	9.9	7.6	11.5	9.7	11.3	10.8	9.5	11.8	12.6	9.6	8.5
SUM	148.7	152.9	157.5	151.7	155.6	142.9	143.9	147.1	144.3	154.8	144.9	140.4	163.8	151.2	147.3	151.4	162.4
Cpk	0.2	0.2	0.3	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Cpm	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Ppk	0.2	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.2	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	Y	Y	N	Y	N	N	N	N	N	N	N	Y	Y	N	N	N	N

```
getChartTable().setTableBackgroundMode(
   SPCGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL);
getChartTable().setBackgroundColor1 (ChartColors.WHITE);
getChartTable().setBackgroundColor2( ChartColors.GRAY);
```

Table and Chart Fonts

There are a large number of fonts that you have control over, both the fonts in the table and the fonts in the chart. The programmer can select a default font (as in the case of non-US character set), or select individual fonts for different elements of the table and charts.

Table Fonts

The table fonts are accessed through the charts **ChartTable** property. Below is a list of accessible table fonts:

Property Name	**Description**
TimeLabelFont	The font used in the display of time values in the table.
SampleLabelFont	The font used in the display of sample numeric values in the table.
CalculatedLabelFont	The font used in the display of calculated values in the table.
StringLabelFont	The font used in the display of header string values in the table.
NotesLabelFont	The font used in the display of notes values in the table.

Extracted from the example BatchVariableControlCharts.BatchIndividualRangeChart

```
this.getChartTable().setSampleLabelFont(new Font("Serif", Font.PLAIN, 12));
```

The **ChartTable** class has a static method **GeneralizedTableDisplay.setDefaultTableFont**, that sets the default Font. Use this if you want to establish a default font for all of the text in a table. This static property must be set BEFORE the charts **Init** routine.

Extracted from the example BatchVariableControlCharts.BatchIndividualRangeChart

```
SPCGeneralizedTableDisplay.setDefaultTableFont(
        new Font("Serif",  Font.PLAIN, 12));
    // Initialize the SPCTimeVariableControlChart
    this.initSPCBatchVariableControlChart(charttype,
            numsamplespersubgroup, numdatapointsinview);
```

Chart Fonts

There are default chart fonts that are static objects in the **SPCChartObjects** class. They establish the default fonts for related chart objects and if you change them they need to be set before the first charts init.. call. Since these properties are static, any changes to them will apply to the program as a whole, not just the immediate class.

AxisLabelFont	The font used to label the x- and y- axes.
AxisTitleFont	The font used for the axes titles.
HeaderFont	The font used for the chart title.
SubheadFont	The font used for the chart subhead.
ToolTipFont	The tool tip font.
AnnotationFont	The annotation font.
ControlLimitLabelFont	The font used to label the control limits

Extracted from the example TimeVariableControlCharts.DynamicXBarRChart

```
SPCChartObjects.setAxisTitleFont ( new Font("Serif", Font.PLAIN, 12));
SPCChartObjects.setControlLimitLabelFont ( new Font("Serif", Font.PLAIN, 10));


this.initSPCTimeVariableControlChart(charttype,
           numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
```

The chart class has a static method, **setDefaultTableFont**, that sets the default Font string. Since the chart fonts all default to different sizes, the default font is defined using a string specifying the name of the font. This static property must be set BEFORE the charts **init** routine.

Extracted from the example Extracted from the example TimeVariableControlCharts.DynamicXBarRChart

```
SPCTimeVariableControlChart.setDefaultChartFontString( "Times");


this.initSPCTimeVariableControlChart(charttype,
           numsamplespersubgroup, numdatapointsinview, timeincrementminutes);.
.
```

Table and Chart Templates

All of the strings displayed in the table and charts use a template unique to the string type. Numeric strings use a **NumericLabel** template, time/date strings use a time **TimeLabel** template, and so on. These templates permit the programmer to customize the display of the strings. Listed below are the various templates.

SPCChartObjects (Accessed in the charts PrimaryChart and SecondaryChart properties)

Propertry	**Type**	**Description**
XValueTemplate	NumericLabel	The x-value template for the data tooltip.
YValueTemplate	NumericLabel	The y-value template for the data tooltip.
XTimeValueTemplate	TimeLabel	x-value template for the data tooltip.
TextTemplate	ChartText	The text template for the data tooltip.

SPCGeneralizedTableDisplay (Accessed in the charts ChartTable property)

Propertry	**Type**	**Description**
TimeItemTemplate	TimeLabel	The TimeLabel object used as a template for displaying time values in the table.
SampleItemTemplate	NumericLabel	The NumericLabel object used as a template for displaying the sample values in the table.
CalculatedItemTemplate	NumericLabel	The NumericLabel object used as a template for displaying calculated values in the table.
StringItemTemplate	StringLabel	The StringLabel object used as a template for displaying string values in the table.
NotesItemTemplate	NotesLabel	The NotesLabel object used as a template for displaying string values in the table.

The most common use for these templates is to set the color attributes of a class of objects, or decimal precision of a numeric string.

```
this.getChartTable().getSampleItemTemplate().setLineColor(ChartColors.RED);
```

Chart Position

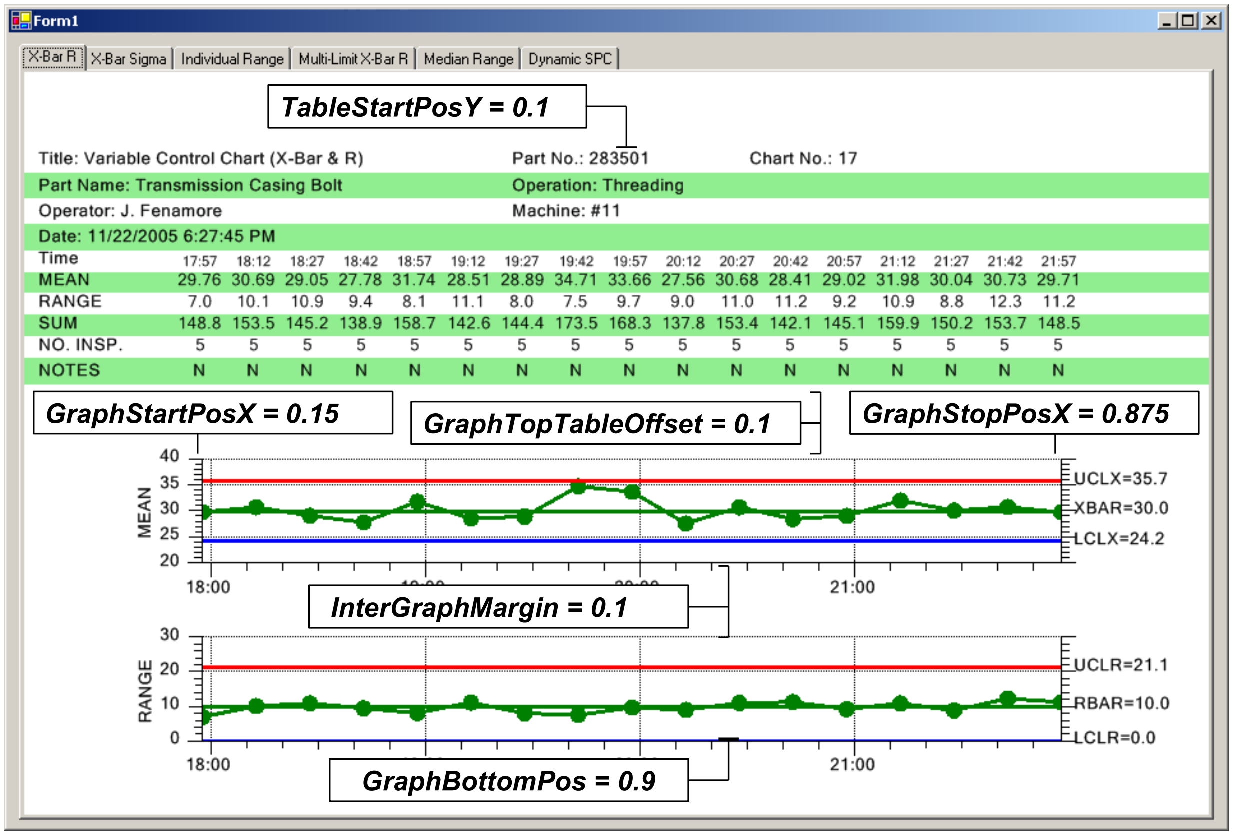
If the SPC chart does not include frequency histograms on the left (they take up about 20% of the available chart width), you may want to adjust the left and right edges of the chart using the **GraphStartPosX** and **GraphStopPlotX** properties to allow for more room in the display of the data. This also affects the table layout, because the table columns must line up with the chart data points.

```
this.setGraphStartPosX( 0.1); // start here
this.setGraphStopPosX( 0.875);  // end here
```

There is not much flexibility positioning the top and bottom of the chart. Depending on the table items enabled, the table starts at the position defined the **TableStartPosY** property, and continues until all of the table items are displayed. It then offsets the top of the primary chart with respect to the bottom of the table by the value of the property **GraphTopTableOffset**. The top of the secondary chart offsets from the bottom of the primary chart by the amount of the property **InterGraphMargin**. The value of the property **GraphBottomPos** defines the bottom of the graph. The default values for these properties are:

```
this.setTableStartPosY( 0.00);
this.setGraphTopTableOffset( 0.02);
this.setInterGraphMargin( 0.075);
this.setGraphBottomPos( 0.925);
```

The picture below uses different values for these properties in order to emphasize the affect that these properties have on the resulting chart.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Fenamore Machine: #11

Date: 11/22/2005 6:27:45 PM

Time	17:57	18:12	18:27	18:42	18:57	19:12	19:27	19:42	19:57	20:12	20:27	20:42	20:57	21:12	21:27	21:42	21:57
MEAN	29.76	30.69	29.05	27.78	31.74	28.51	28.89	34.71	33.66	27.56	30.68	28.41	29.02	31.98	30.04	30.73	29.71
RANGE	7.0	10.1	10.9	9.4	8.1	11.1	8.0	7.5	9.7	9.0	11.0	11.2	9.2	10.9	8.8	12.3	11.2
SUM	148.8	153.5	145.2	138.9	158.7	142.6	144.4	173.5	168.3	137.8	153.4	142.1	145.1	159.9	150.2	153.7	148.5
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

SPC Control Limits

There are two ways to set the SPC control limit for a chart. The first explicitly sets the limits to values that you calculate on your own, because of some analysis that a quality engineer does on previously collected data. The second auto-calculates the limits using the algorithms supplied in this software.

The quick way to set the limit values and limit strings is to use the charts **getChartData().setControlLimitValues** and **getChartData().setControlLimitStrings** methods. This method only works for the default +-3-sigma level control limits, and not any others you may have added using the charts **addAdditionalControlLimit** method discussed in the *Multiple Control Limits* section. The data values in the *controllimitvalues* and *controllimitstrings* arrays used to pass the control limit information must be sorted in the following order:

[SPC_PRIMARY_CONTROL_TARGET,
SPC_PRIMARY_LOWER_CONTROL_LIMIT,
SPC_PRIMARY_UPPER_CONTROL_LIMIT,
SPC_SECONDARY_CONTROL_TARGET,
SPC_SECONDARY_LOWER_CONTROL_LIMIT,
SPC_SECONDARY_UPPER_CONTROL_LIMIT]

Example code extracted from the TimeVariableControlsCharts.MedianRangeChart example program.

```
SPCControlChartData chartdata = this.getChartData();

double [] controllimitvalues = {30, 24, 36, 10, 0, 22};
chartdata.setControlLimitValues(controllimitvalues);

String [] controllimitstrings = {"XBAR","LCL", "UCL","RBAR","LCL","UCL"};
chartdata.setControlLimitStrings(controllimitstrings);
```

You can also set the control limit values and control limit text one value at a time using the **getChartData().setControlLimitValue** and **getChartData().setControlLimitString** methods.

A more complicated way to set the control limits explicitly is to first grab a reference to the **SPCControlLimitRecord** for a given control limit, and then change the value of that control limit, and the control limit text, if desired. The example below sets the control limit values and text for the three control limits (target value, upper control limit, and lower control limit) of the primary chart, and the three control limit values for the secondary chart.

```
//target control limit primary chart
SPCControlLimitRecord primarytarget=
chartdata.getControlLimitRecord(SPCControlChartData.SPC_PRIMARY_CONTROL_TARGET);
primarytarget.setControlLimitValue ( 30);
primarytarget.setControlLimitText ( "XBAR");

//lower control limit primary chart
SPCControlLimitRecord primarylowercontrollimit =

chartdata.getControlLimitRecord(SPCControlChartDat
a.SPC_PRIMARY_LOWER_CONTROL_LIMIT);
primarylowercontrollimit.setControlLimitValue ( 24);
primarylowercontrollimit.setControlLimitText ( "LCL");

//upper control limit primary chart
SPCControlLimitRecord primaryuppercontrollimit =
chartdata.getControlLimitRecord(SPCControlChartDat
a.SPC_PRIMARY_UPPER_CONTROL_LIMIT);
primaryuppercontrollimit.setControlLimitValue ( 36);
primaryuppercontrollimit.setControlLimitText ( "UCL");
```

```
// Set control limits for secondary chart

//target control limit secondary chart
SPCControlLimitRecord secondarytarget =
chartdata.getControlLimitRecord(SPCControlChartData.SPC_SECONDARY_CONTROL_TARGET);
secondarytarget.setControlLimitValue ( 10);
secondarytarget.setControlLimitText ( "RBAR");

//lower control limit secondary chart
SPCControlLimitRecord secondarylowercontrollimit =
chartdata.getControlLimitRecord(SPCControlChartDat
a.SPC_SECONDARY_LOWER_CONTROL_LIMIT);
secondarylowercontrollimit.setControlLimitValue ( 0);
secondarylowercontrollimit.setControlLimitText ( "LCL");

//upper control limit secondary chart
SPCControlLimitRecord secondaryuppercontrollimit =
chartdata.getControlLimitRecord(SPCControlChartDat
a.SPC_SECONDARY_UPPER_CONTROL_LIMIT);
secondaryuppercontrollimit.setControlLimitValue ( 22);
secondaryuppercontrollimit.setControlLimitText ( "UCL");
```

The second way to set the control limits is to call the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

```
// Must have data loaded before any of the Auto.. methods are called
simulateData();

// Calculate the SPC control limits for both graphs of the current SPC

this.autoCalculateControlLimits ();
```

You can add data to the **ChartData** object, auto-calculate the control limits to establish the SPC control limits, and continue to add new data values. Alternatively, you can set the SPC control limits explicitly, as the result of previous runs, using the previously described **getChartData().setControlLimitValues** method, add new sampled data values to the **ChartData** object, and after a certain number of updates call the **autoCalculateControlLimits** method to establish new control limits.

```
updateCount++;
```

```
this.getChartData().addNewSampleRecord (timestamp, samples);
if (updateCount > 50) // After 50 sample groups and calculate limits on the fly
{
// Calculate the SPC control limits for the X-Bar part of the current SPC chart
    this.autoCalculateControlLimits ();
    // Scale the y-axis of the X-Bar chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    this.autoScaleSecondaryChartYRange();
}
```

The **autoCalculateControlLimits** method calculates the control limits for both the primary and secondary charts. If you want to auto-calculate the control limits for just one of the charts, call the **autoCalculatePrimaryControlLimits** or **autoCalculateSecondaryControlLimits** method.

Need to exclude records from the control limit calculation? Call the **getChartData().excludeRecordFromControlLimitCalculations** method, passing in true to exclude the record.

```
for (int i=0; i < 10; i++)
      this.getChartData().excludeRecordFromControlLimitCalculations(i,true);
```

Formulas Used in Calculating Control Limits for Variable Control Charts

The SPC control limit formulas used by **autoCalculateControlLimits** in the software derive from the following sources:

X-Bar R, X-Bar Sigma, EWMA, MA and CuSum - "Introduction to Statistical Quality Control" by Douglas C. Montgomery, John Wiley and Sons, Inc. 2001.

Median-Range, Individual-Range - "SPC Simplified – Practical Steps to Quality" by Robert T. Amsden, Productivity Inc., 1998.

SPC Control Chart Nomenclature

UCL = Upper Control Limit

LCL = Lower Control Limit

Center line = The target value for the process

$\overline{\overline{X}}$ = X double-bar - Mean of sample subgroup means (also called the grand average)

$\overline{R}$ = R-bar – Mean of sample subgroup ranges

$\tilde{R}$ = R-Median – Median of sample subgroup ranges

S = Sigma – sample standard deviation

$\overline{S}$ = Sigma-bar – Average of sample subgroup sigma's

M = sample Median

$\tilde{M}$ = Median of sample subgroup medians

X-Bar R Chart – Also known as the Mean (or Average) and Range Chart

Control Limits for the X-Bar Chart

$$\text{UCL} = \overline{\overline{X}} + A_2 * \overline{R}$$

$$\text{Center line} = \overline{\overline{X}}$$

$$\text{LCL} = \overline{\overline{X}} - A_2 * \overline{R}$$

Control Limits for the R-Chart

$$\text{UCL} = \overline{R} + D_4 * \overline{R}$$

$$\text{Center line} = \overline{R}$$

$$\text{LCL} = \overline{R} - D_3 * \overline{R}$$

Where the constants A_2, D_3 and D_4 are tabulated in every SPC textbook for various sample sizes.

X-Bar Sigma – Also known as the X-Bar S Chart

Control Limits for the X-Bar Chart

$$\text{UCL} = \bar{\bar{X}} + A_3 * \bar{S}$$

$$\text{Center line} = \bar{\bar{X}}$$

$$\text{LCL} = \bar{\bar{X}} - A_3 * \bar{S}$$

Control Limits for the Sigma-Chart

$$\text{UCL} = \overline{B}_4 * \bar{S}$$

$$\text{Center line} = \bar{S}$$

$$\text{LCL} = \overline{B}_3 * \bar{S}$$

Where the constants A_3, B_3 and B_4 are tabulated in every SPC textbook for various sample sizes.

Median Range – Also known as the Median and Range Chart

Control Limits for the Median Chart

$$\text{UCL} = \tilde{M} + \tilde{A}_2 * \tilde{R}$$

$$\text{Center line} = \tilde{M}$$

$$\text{LCL} = \tilde{M} - \tilde{A}_2 * \tilde{R}$$

Control Limits for the R-Chart

$$\text{UCL} = \tilde{R} + \tilde{D}_4 * \tilde{R}$$

$$\text{Center line} = \tilde{R}$$

$$\text{LCL} = \tilde{R} - \tilde{D}_3 * \tilde{R}$$

The constants A_2, D_3 and D_4 for median-range charts are different than those for mean-range charts. A brief tabulation of the median-range chart specific values appears below

Size	**A2**	**D3**	**D4**
2	2.22	0.0	3.87
3	1.26	0.0	2.75
4	0.83	0.0	2.38
5	0.71	0.0	2.18

Individual Range Chart – Also known as the X-R Chart

Control Limits for the X-Bar Chart

$$\text{UCL} = \bar{X} + E_2 * \bar{R}$$

$$\text{Center line} = \overset{=}{X}$$

$$\text{LCL} = \bar{X} - E_2 * \bar{R}$$

Control Limits for the R-Chart

$$\text{UCL} = \bar{R} + D_4 * \bar{R}$$

$$\text{Center line} = \bar{R}$$

$$\text{LCL} = 0$$

$\bar{R}$ in this case is the average of the moving ranges.

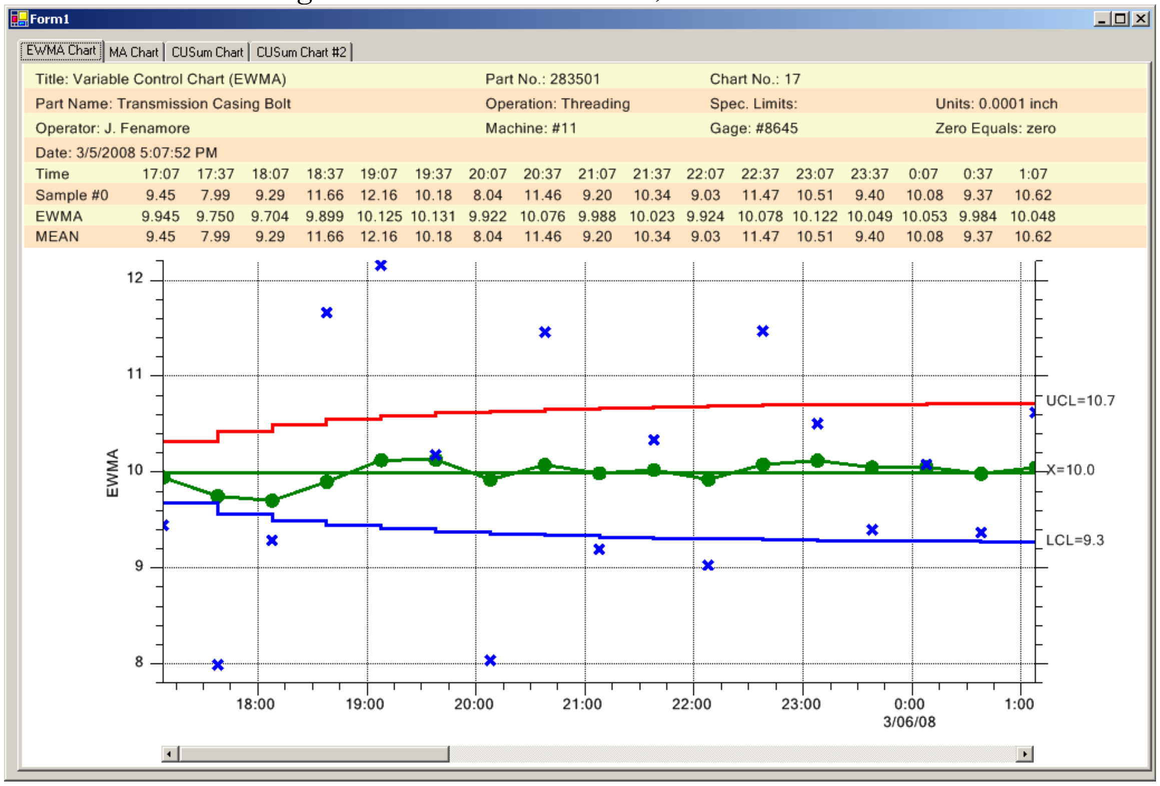
–

X in this case is the mean of the samples

Where the constants E_2 and D_4 are tabulated in every SPC textbook for various sample sizes.

EWMA Chart – Exponentially Weighted Moving Average

A EWMA chart showing the variable control limits, actual values and EWMA values



Title: Variable Control Chart (EWMA)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation: Threading	Spec. Limits:	Units: 0.0001 inch
Operator: J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 3/5/2008 5:07:52 PM			

Time	17:07	17:37	18:07	18:37	19:07	19:37	20:07	20:37	21:07	21:37	22:07	22:37	23:07	23:37	0:07	0:37	1:07
Sample #0	9.45	7.99	9.29	11.66	12.16	10.18	8.04	11.46	9.20	10.34	9.03	11.47	10.51	9.40	10.08	9.37	10.62
EWMA	9.945	9.750	9.704	9.899	10.125	10.131	9.922	10.076	9.988	10.023	9.924	10.078	10.122	10.049	10.053	9.984	10.048
MEAN	9.45	7.99	9.29	11.66	12.16	10.18	8.04	11.46	9.20	10.34	9.03	11.47	10.51	9.40	10.08	9.37	10.62

The current value (z) for an EWMA chart is calculated as an exponentially weighted moving average of all previous samples.

$z_i =$ Caps Lock $* x_i + (1 -$ Caps Lock $) z_{i-1}$

where x_i is the sample value for time interval i, the smoothing value Caps Lock has the permissible range of 0 < Caps Lock 1 and the starting value (required with the first sample at i = 0) is the process target value, F7 $_0$.

Control Limits for the EWMA Chart

UCL = F7 $_0$ + L * Shift Sqrt(((Caps Lock/(2- .)) * (1- (1- .)^{2i}))

Center line $= \mu_0$

LCL $= \mu_0 - L * \sigma \mathrm{Sqrt}(((\lambda/(2-\lambda)) * (1-(1-\lambda)^{2i}))$

μ_0 is the process mean

σ is the process standard deviation, also known as sigma

λ is the user specified smoothing value. A typical value for λ is 0.05, 0.1 or 0.2

L is the width of the control limits. The typical value for L is in the range of 2.7 to 3.0 (corresponding to the usual three-sigma control limits).

The software does not calculate optimal λ and L values; that is up to you, the programmer to supply, based on your experience with EWMA charts.

Note that the term $(1-(1-\lambda)^{2i})$ approaches unity as i increases. The implies that the control limits of an EWMA chart will reach approximate steady state values defined by:

UCL $= \mu_0 + L * \sigma \mathrm{Sqrt}(\lambda/(2-\lambda))$

LCL $= \mu_0 - L * \sigma \mathrm{Sqrt}(\lambda/(2-\lambda))$

It is best if you use the exact equations that take into account the sample period, so that an out of control process can be detected using the tighter control limits that are calculated for small i.

If the EWMA chart is used with subgroup sample sizes greater than 1, the value of x_i is replace by the mean of the corresponding sample subgroup, and the value of σ is replaced by the value σ/sqrt(n), where in is the sample subgroup size.

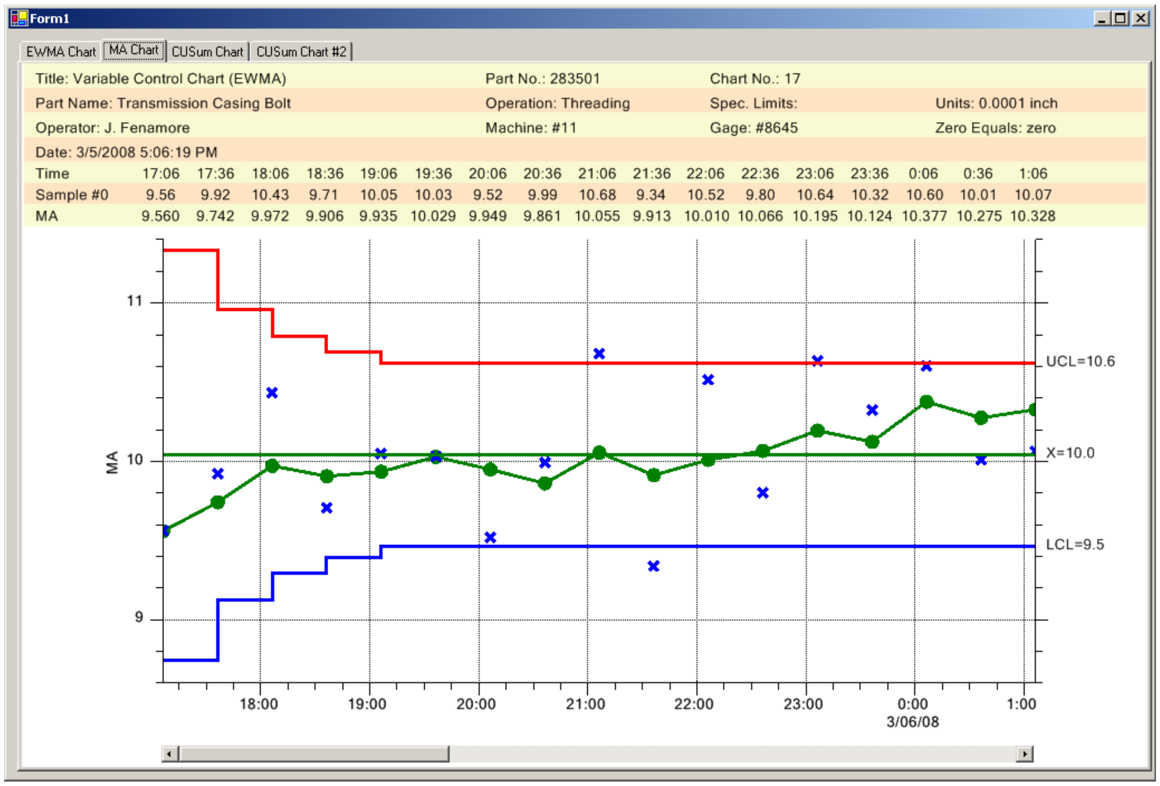
You specify λ and L immediately after the initialization call **initSPCTimeVaraibleControlChart** (for a time-based variable control chart), or **initSPCBatchVariableControlChart** (for a batch-based variable control chart). See the examples MiscTimeBasedControlCharts.EWMAChart, and MiscBatchBasedControlCharts.EWMAChart. Specify L using the **setDefaultControlLimitSigma** method, and λ using the **setEWMA_Lambda** method. You can optionally set the EWMA starting value (**EWMA**_StartingValue), normally set to the process mean value, and whether or not to use the steady-state EWMA control limits (**useSSLimits**).

Extracted from the MiscTimeBasedControlCharts.EWMAChart example.

```
//  SPC variable control chart type
int charttype = SPCControlChartData.EWMA_CHART;
.
.
.
// Initialize the SPCTimeVariableControlChart
this.initSPCTimeVariableControlChart(charttype,
    numsamplespersubgroup, numdatapointsinview, sampleincrement);

SPCControlChartData chartdata = this.getChartData();

chartdata.setEWMA_StartingValue(10); // estimate of mean of process variable.
chartdata.setEWMA_Lambda ( 0.1);
this.setDefaultControlLimitSigma ( 2.7); // Specifies L value
chartdata.setEWMA_UseSSLimits( false);
```

MA Chart – Moving Average

A MA chart showing the variable control limits, actual values and moving average values



Title: Variable Control Chart (EWMA) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch

Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero

Date: 3/5/2008 5:06:19 PM

Time	17:06	17:36	18:06	18:36	19:06	19:36	20:06	20:36	21:06	21:36	22:06	22:36	23:06	23:36	0:06	0:36	1:06
Sample #0	9.56	9.92	10.43	9.71	10.05	10.03	9.52	9.99	10.68	9.34	10.52	9.80	10.64	10.32	10.60	10.01	10.07
MA	9.560	9.742	9.972	9.906	9.935	10.029	9.949	9.861	10.055	9.913	10.010	10.066	10.195	10.124	10.377	10.275	10.328

The current value (z) for a MA chart is calculated as a weighted moving average of the N most recent samples.

$$z_i = (x_i + x_{i-1} + x_{i-2} + \dots \ x_{i-N+1})/N$$

where x_i is the sample value for time interval i, and N is the length of the moving average.

Control Limits for the MA Chart

UCL = $\boxed{F7}_0$ + 3 * $\boxed{Shift}$/sqrt(N)

Center line = $\boxed{F7}_0$

LCL = $\boxed{F7}_0$ - 3 * $\boxed{Shift}$/sqrt(N)

$\boxed{F7}_0$ is the process mean

is the process standard deviation, also known as sigma

N is the length of the moving average used to calculate the current chart value

Control Limits for the MR part of the MAMR (Moving Average/Moving Range Chart

Control Limits for the R-Chart

UCL	=	$\bar{R} + D_4 * \bar{R}$
Center line	=	$\bar{R}$
LCL	=	0

$\bar{R}$ in this case is the average of the moving ranges.

Where the constant D_4 is tabulated in every SPC textbook for various sample sizes.

Control Limits for the MS part of the MAMS (Moving Average/Moving Sigma Chart

UCL	=	$\bar{B}_4 * \bar{S}$
Center line	=	$\bar{S}$
LCL	=	$\bar{B}_3 * \bar{S}$

$\bar{S}$ in this case is the average of the moving sigmas.

Where the constant B_4 is tabulated in every SPC textbook for various sample sizes.
The software does not calculate an optimal N value; that is up to you, the programmer to supply, based on your past experience with MA charts.

For the values of z_i where $i < N-1$, the weighted average and control limits are calculated using the actual number of samples used in the average, rather than N. This results in expanded values for the control limits for small $i < N-1$.

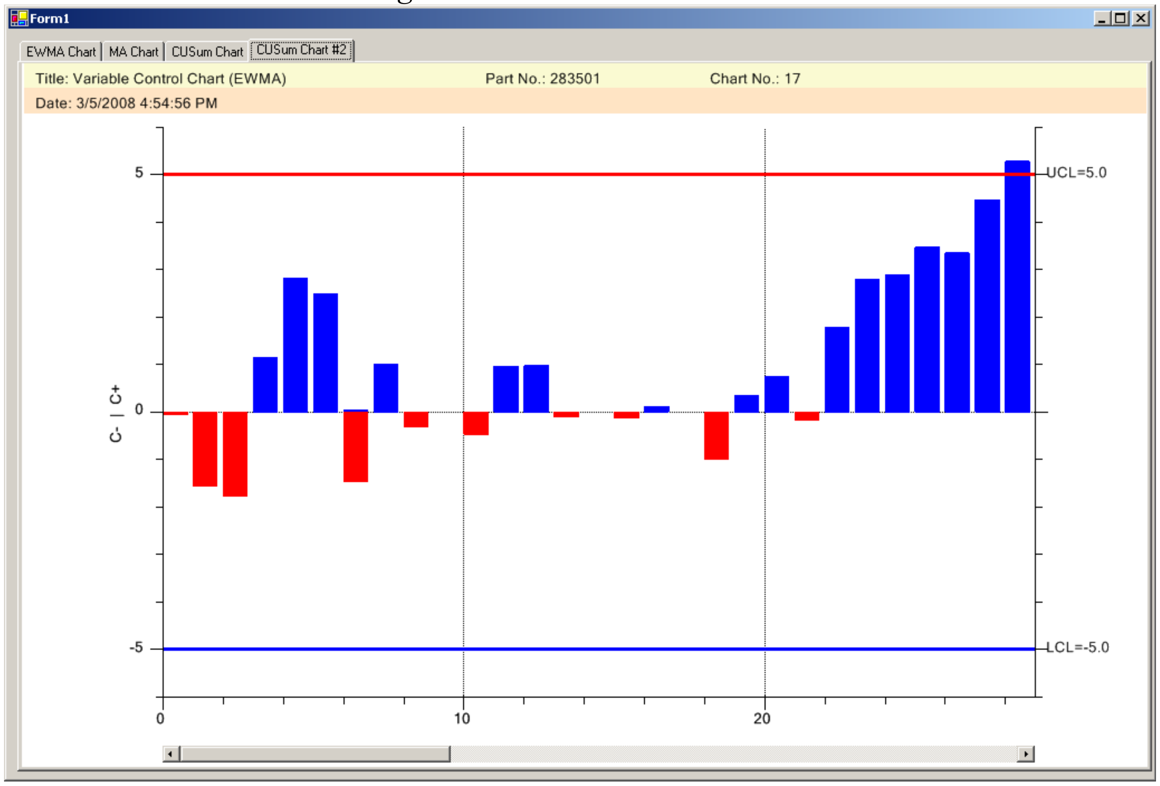
You specify N, the length of the moving average, immediately after the initialization call **initSPCTimeVaraibleControlChart** (for a time-based variable control chart), or **initSPCBatchVariableControlChart** (for a batch-based variable control chart). Set the process mean and process sigma used in the control limit calculations using the **setProcessMean** and **setProcessSigma** methods.
See the examples MiscTimeBasedControlCharts.MAChart, and MiscBatchBasedControlCharts.MAChart. Specify N using the **setMA_w** property.

Extracted from the MiscTimeBasedControlCharts.MAChart example.

```
//  SPC variable control chart type
int charttype = SPCControlChartData.MA_CHART;
.
.
.
// Initialize the SPCTimeVariableControlChart
this.initSPCTimeVariableControlChart(charttype,
numsamplespersubgroup, numdatapointsinview, sampleincrement);
// Number of time periods in moving average
this.getChartData().setMA_w( 9);
```

CuSum Chart – Tabular, one-sided, upper and lower cumulative sum

A batch CuSum chart exceeding the H value



Form1

EWMA Chart | MA Chart | CUSum Chart | CUSum Chart #2

Title: Variable Control Chart (EWMA) Part No.: 283501 Chart No.: 17

Date: 3/5/2008 4:54:56 PM

The tabular cusum works by accumulating deviations from the process mean, $\boxed{F7}_0$. Positive deviations are accumulated in the one sided upper cusum statistic, C^+, and negative deviations are accumulated in the one sided lower cusum statistic, C^-. The statistics are calculated using the following equations:

$$C^+_i = \max[0, x_i - (\boxed{F7}_0 + K) + C^+_{i-1}]$$

$$C^-_i = \max[0, (\boxed{F7}_0 - K) - x_i + C^+_{i-1}]$$

where the starting values are $C^+_0 = C^-_0 = 0$

$\boxed{F7}_0$ is the process mean

K is the reference (or slack value) that is usually selected to be one-half the magnitude of the difference between the target value, $\boxed{F7}_0$, and the out of control process mean value, $\boxed{F7}_1$, that you are trying to detect.

$K = ABS(\boxed{F7}_1 - \boxed{F7}_0)/2$

The control limits used by the chart are +-H. If the value of either C^+ or C^- exceed +- H, the process is considered out of control.

The software does not calculate an optimal H or K value; that is up to you, the programmer to supply, based on your past experience with CuSum charts. Typically H is set equal to 5 times the process standard deviation, Shift. Typically K is selected to be one-half the magnitude of the difference between the target value, F7$_0$, and the out of control process mean value, F7$_1$, that you are trying to detect. You specify H and K in the initialization call **initSPCTimeCusumControlChart** (for a time-based variable control chart), or **initSPCBatchCusumControlChart** (for a batch-based variable control chart). See the examples MiscTimeBasedControlCharts.CUSumChart, MiscTimeBasedControlCharts.CUSumChart2, MiscBatchBasedControlCharts.CUSumChart, and MiscBatchBasedControlCharts.CUSumChart2.

Extracted from MiscTimeBasedControlCharts.CUSumChart

```
int charttype = SPCControlChartData.TABCUSUM_CHART;
double processMean = 10;
double kValue = 0.5;
double hValue = 5;
.
.
// Initialize the SPCTimeVariableControlChart
this.initSPCTimeCusumControlChart(charttype,
    numsamplespersubgroup, numdatapointsinview, sampleincrement,
    processMean, kValue, hValue);
```

Or, you can call the **initSPCTimeCusumControlChart** method and specify H and K using immediately afterwards using simple property calls.

Extracted from MiscTimeBasedControlCharts.CUSumChart2

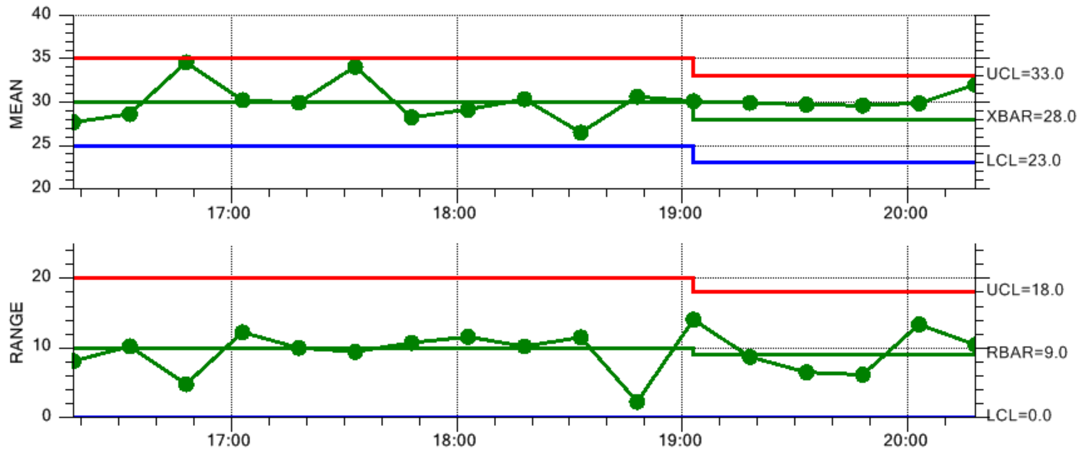
```
int charttype = SPCControlChartData.TABCUSUM_CHART;
double processMean = 10;
double kValue = 0.5;
double hValue = 5;
.
.
.
// Initialize the SPCTimeVariableControlChart
this.initSPCTimeVariableControlChart(charttype,
    numsamplespersubgroup, numdatapointsinview, sampleincrement);
this.getChartData().setCusumHValue( hValue);
this. getChartData().setCusumKValue(kValue);
this. getChartData().setProcessMean( processMean);
```

Variable SPC Control Limits

There can be situations where SPC control limits change in a chart. If the control limits change, you need to set the following **ControlLineMode** property to SPCChartObjects.CONTROL_LINE_VARIABLE, as in the example below. The default value is SPCChartObjects.CONTROL_LINE_FIXED.

```
this.getPrimaryChart().setControlLineMode( SPCChartObjects.CONTROL_LINE_VARIABLE);
this.getSecondaryChart().setControlLineMode(SPCChartObject
s.CONTROL_LINE_VARIABLE);
```

In the SPCChartObjects.CONTROL_LINE_FIXED case, the current SPC control limit plots as a horizontal straight line for the entire width of the chart, regardless if the control limit changes, either explicity, or using the **autoCalculateControlLimits** method. If the **ControlLineMode** property is SPCChartObjects.CONTROL_LINE_VARIABLE, the SPC limit value plots at the value it had when the sample subgroup values updated. If you change a control limit value, the control limit line will no longer be a straight horizontal line, instead it will be jagged, or stepped, depending on the changes made.



There are three ways to enter new SPC limit values. See the example program TimeVaraibleControlCharts.VariableControlLimits for an example of all three methods. First, you can use the method **getChartData().setControlLimitValues** method.

```
double [] initialControlLimits = {30, 25, 35, 10, 0, 20};
double [] changeControlLimits = {28, 23, 33, 9, 0, 18};
.
.
this.getPrimaryChart().setControlLineMode(SPCChartObjects.CONTROL_LINE_VARIABLE);
.
.
// Change limits at sample subgroup 10
if (i== 10)
{
    this.getChartData().setControlLimitValues (changeControlLimits);
}
this.getChartData().addNewSampleRecord (timestamp, samples, notesstring);
```

Second, you can use the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

```
.
.
this.getPrimaryChart().setControlLineMode( SPCChartObjects.CONTROL_LINE_VARIABLE);
.
.

//  Variable Control Limits re-calculated every update after 10 using
//  autoCalculateControlLimits
    if (i > 10)
        this.autoCalculateControlLimits ();
    this.getChartData().addNewSampleRecord (timestamp, samples, notesstring);
```

Last, you can enter the SPC control limits with every new sample subgroup record, using one of the methods that include a control limits array parameter.

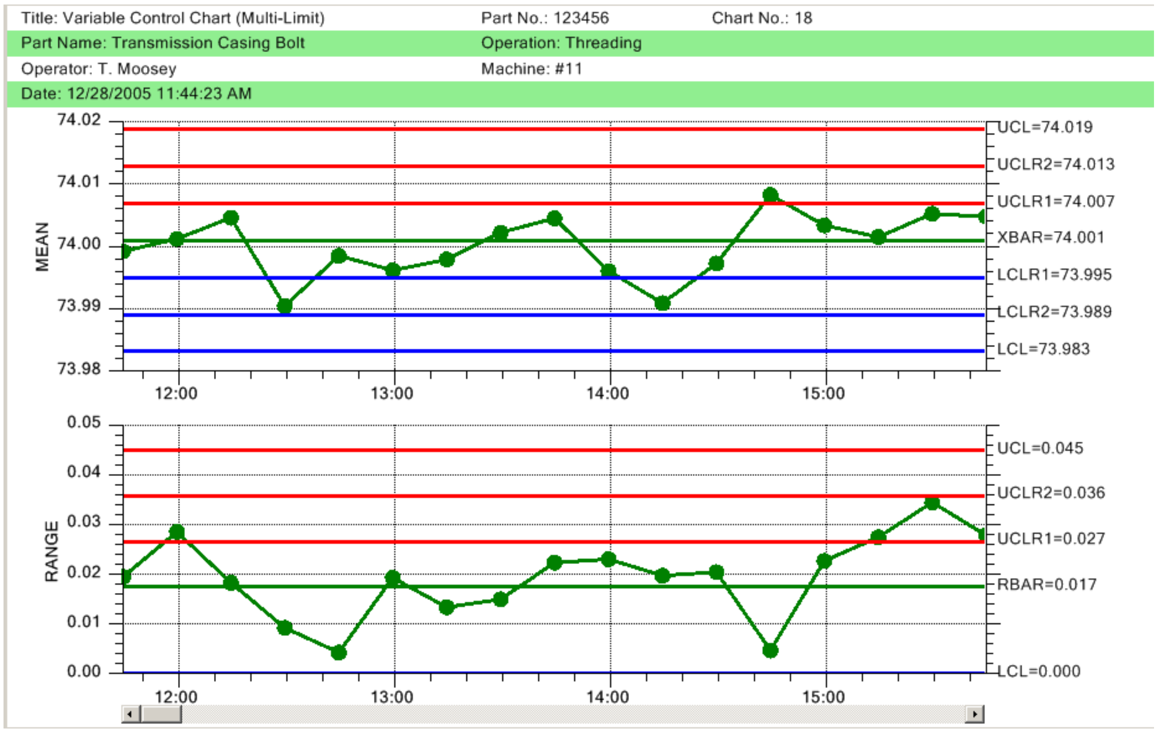
```
double [] initialControlLimits = {30, 25, 35, 10, 0, 20};
double [] changeControlLimits = {28, 23, 33, 9, 0, 18};
DoubleArray variableControlLimits;
.
.
this.getPrimaryChart().setControlLineMode(SPCChartObjects.CONTROL_LINE_VARIABLE);
```

```
.
.
//      Variable Control Limits updated using addNewSampleRecord
    if (i== 10) // need to convert changeControlLimits to a DoubleArray
        variableControlLimits = new DoubleArray(changeControlLimits);
    this.getChartData().addNewSampleRecord (timestamp, samples,
        variableControlLimits, notesstring);
```

Multiple SPC Control Limits

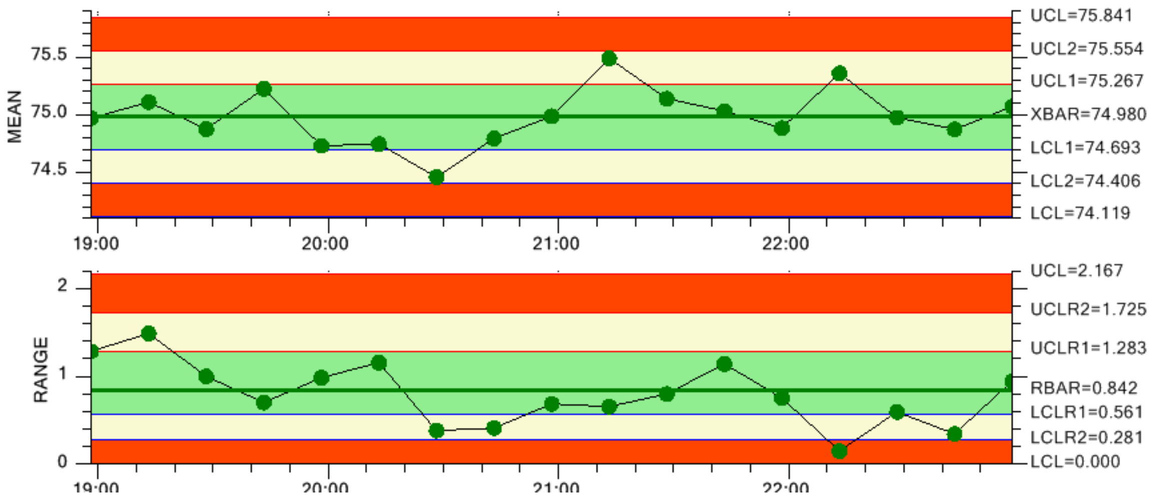
The normal SPC control limit displays at the 3-sigma level, both high and low. A common standard is that if the process variable under observation falls outside of the +-3-sigma limits the process is out of control. The default setup of our variable control charts have a high limit at the +3-sigma level, a low limit at the -3-sigma levet, and a target value. There are situations where the quality engineer also wants to display control limits at the 1-sigma and 2-sigma level. The operator might receive some sort of preliminary warning if the process variable exceeds a 2-sigma limit.

You are able to add additional control limit lines to a variable control chart, as in the example program TimeVariableControlCharts.MultiLimitXBarRChart.



We also added a method (add3SigmaControlLimits) which will generate multiple control limits, for +-1, 2, and 3- sigma levels, based on an initial specification of the target value, and the +-3 sigma control limits. This is most useful if you want to generate +-1, 2 and 3-sigma control limits in order to fill in between them with a zone fill color. See the TimeVariableControlCharts.MultiLimitXBarRChart example. If you call the AutoCalculateControlLimits method, the initial +-1,2 and 3-sigma control limit values will be altered to the new, calculated values, but the control limit lines remain, with their new values. Since you do not normally want to be generating alarm messages for excursions into the +-1 and 2-sigma limit areas, the add3SigmaControl limits has the option of disabling alarm notification in the case of +-1 and +-2 alarm conditions.

```
double target = 75, lowlim = 74, highlim = 76;

boolean limitcheck = false;

this.getPrimaryChart().add3SigmaControlLimits(target, lowlim, highlim,
limitcheck);

this.getPrimaryChart().setControlLimitLineFillMode(true);


target = 1; lowlim = 0; highlim = 2;

this.getSecondaryChart().add3SigmaControlLimits(target, lowlim, highlim,
limitcheck);

this.getSecondaryChart().setControlLimitLineFillMode(true);
```



Control Limit Fill Option used with +-1, 2 and 3-sigma control limits

You can also add additional control limits one at a time. By default you get the +-3-sigma control limits. So additional control limits should be considered +-2-sigma and +-1-sigma control limits. Do not confuse control limits with specification limits, which must be added using the **addSpecLimit** method. There are two steps to adding additional control limits: creating a SPCControlLimitRecord object for the new control limit, and adding the control limit to the chart using the charts addAdditionalControlLimit method. It is critical that you add them in a specific order, that order being:

Primary Chart	SPC_LOWER_CONTROL_LIMIT_2	(2-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_2	(2-sigma lower limit)
Primary Chart	SPC_LOWER_CONTROL_LIMIT_1	(1-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_1	(1-sigma lower limit)
Secondary Chart	SPC_LOWER_CONTROL_LIMIT_2	(2-sigma lower limit)
Secondary Chart	SPC_UPPER_CONTROL_LIMIT_2	(2-sigma lower limit)
Secondary Chart	SPC_LOWER_CONTROL_LIMIT_1	(1-sigma lower limit)
Secondary Chart	SPC_UPPER_CONTROL_LIMIT_1	(1-sigma lower limit)

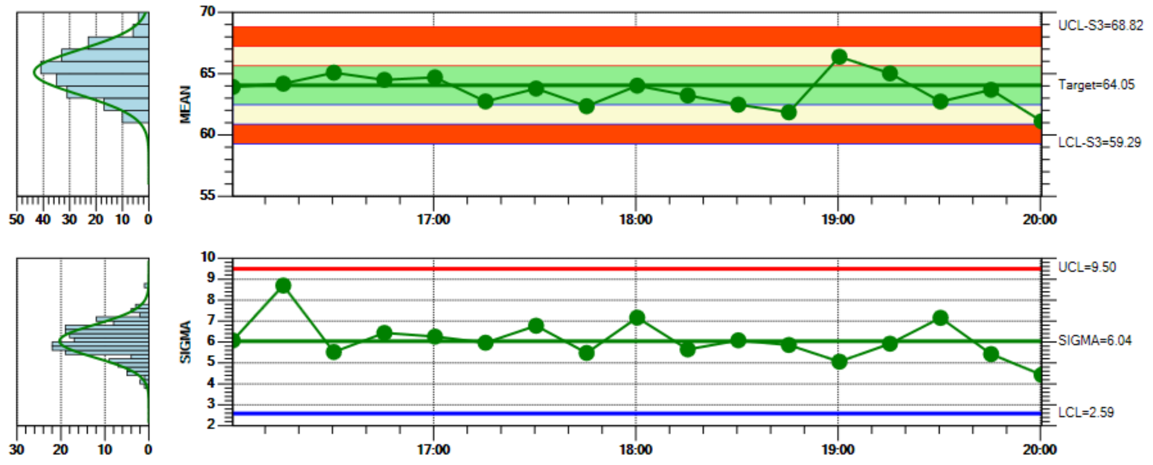
```
double sigma2 = 2.0;

double sigma1 = 1.0;

// Create multiple limits

// For PrimaryChart

SPCControlLimitRecord  lcl2 = new SPCControlLimitRecord(this.getChartData(),
SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0,"LCLR2", "LCLR2");

SPCControlLimitRecord  ucl2 = new SPCControlLimitRecord(this.getChartData(),
SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 0,"UCLR2", "UCLR2");


primaryChart.addAdditionalControlLimit(lcl2,
SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_2, sigma2);

primaryChart.addAdditionalControlLimit(ucl2,
SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);


SPCControlLimitRecord  lcl3 = new SPCControlLimitRecord(this.getChartData(),
SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0,"LCLR1", "LCLR1");

SPCControlLimitRecord  ucl3 = new SPCControlLimitRecord(this.getChartData(),
SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 0,"UCLR1", "UCLR1");


primaryChart.addAdditionalControlLimit(lcl3,
SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_1, sigma1);

primaryChart.addAdditionalControlLimit(ucl3,
SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);


// For SecondarChart - high limits only

SPCControlLimitRecord  ucl4 = new SPCControlLimitRecord(this.getChartData(),
SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 0,"UCLR2", "UCLR2");

SPCControlLimitRecord  ucl5 = new SPCControlLimitRecord(this.getChartData(),
SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 0,"UCLR1", "UCLR1");


secondaryChart.addAdditionalControlLimit(ucl4,
SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);

secondaryChart.addAdditionalControlLimit(ucl5,
SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);
```

Special Note – When you create a **SPCControlLimitRecord** object, you can specifify an actual limit level. If you do not call the charts **autoCalculateControlLimits** method, the control limit will be displayed at that value. If you do call **autoCalculateControlLimits** method, the auto-calculated value overrides the initial value (0.0 in the examples above). When you call the charts **addAdditionalControlLimit s** method, you specify the sigma level that is used by the **autoCalculateControlLimits** to calculate the control limit level.

If you want the control limits displayed as filled areas, set the charts **ControlLimitLineFillMode** property True.

```
this.getPrimaryChart().setControlLimitLineFillMode(true);
```

This will fill each control limit line from the limit line to the target value of the chart. In order for the fill to work properly, you must set this property after you define all additional control limits. Also, you must add the outer most control limits (SPC_UPPER_CONTROL_LIMIT_3 and SPC_LOWER_CONTROL_LIMIT_3) first, followed by the next outer most limits (SPC_UPPER_CONTROL_LIMIT_2 and SPC_LOWER_CONTROL_LIMIT_2), followed by the inner most control limits (SPC_UPPER_CONTROL_LIMIT_1 and SPC_LOWER_CONTROL_LIMIT_1). This way the fill of the inner limits will partially cover the fill of the outer limits, creating the familiar striped look you want to see.



Western Electric (WE) Runtime Rules

The normal SPC control limit rules display at the 3-sigma level, both high and low. In this case, a simple threshold test determines if a process is in, or out of control. Other, more complex tests rely on more complicated decision-making criteria. The most popular of these are the Western Electric Rules, also know as the WE Rules, or WE Runtime Rules. These rules utilize historical data for the eight most recent sample intervals and look for a non-random pattern that can signify that the process is out of control, before reaching the normal +-3 sigma limits. A processed is considered out of control if any of the following criteria are met:

*Starting with Revision 3.0, we have added additional, industry standard rules to the software: Nelson, AAIG, Juran, Hughes, Gitlow, Westgard, and Duncan. In addition, you can mix and match rules from the different rule sets, and you can create your own custom rules using our standardize rule templates. See Chapter 8, for more information.

1. **The most recent point plots outside one of the 3-sigma control limits.** If a point lies outside either of these limits, there is only a 0.3% chance that this was caused by the normal process.

2. **Two of the three most recent points plot outside and on the same side as one of the 2-sigma control limits.** The probability that any point will fall outside the warning limit is only 5%. The chances that two out of three points in a row fall outside the warning limit is only about 1%.

3. **Four of the five most recent points plot outside and on the same side as one of the 1-sigma control limits.** In normal processing, 68% of points fall within one sigma of the mean, and 32% fall outside it. The probability that 4 of 5 points fall outside of one sigma is only about 3%.

4. **Eight out of the last eight points plot on the same side of the center line, or target value.** Sometimes you see this as 9 out of 9, or 7 out of 7. There is an equal chance that any given point will fall above or below the mean. The chances that a point falls on the same side of the mean as the one before it is one in two. The odds that the next point will also fall on the same side of the mean is one in four. The probability of getting eight points on the same side of the mean is only around 1%.

These rules apply to both sides of the center line at a time. Therefore, there are eight actual alarm conditions: four for the above center line sigma levels and four for the below center line sigma levels.

There are also additional WE Rules for trending. These are often referred to as WE Supplemental Rules. Don't rely on the rule number, often these are listed in a different order.

5. **Six points in a row increasing or decreasing.** The same logic is used here as for rule 4 above. Sometimes this rule is changed to seven points rising or falling.

6. **Fifteen points in a row within one sigma.** In normal operation, 68% of points will fall within one sigma of the mean. The probability that 15 points in a row will do so, is less than 1%.

7. **Fourteen points in a row alternating direction.** The chances that the second point is always higher than (or always lower than) the preceding point, for all seven pairs is only about 1%.

8. **Eight points in a row outside one sigma.** Since 68% of points lie within one sigma of the mean, the probability that eight points in a row fall outside of the one-sigma line is less than 1%.

While the techniques in the previous section can be used to draw multiple SPC control limit lines on the graph, at the +-1, 2, 3 sigma levels for example, they do not provide for the (x out of y) control criteria used in evaluating the WE rules. The software can be explicitly flagged to evaluate out of control alarm conditions according to the WE Rules, instead of the default +-3 sigma control criteria. It will create alarm lines at the +-1, 2, and 3-sigma control limits and the center line. It will also automatically establish the eight alarm conditions associated with the WE rules. Set the WE rules flag using the PrimaryChart (or SecondaryChart) **useWERuntimeRules** method. When the variable control charts **autoCalculatedControlLimits** method is called, the software automatically calculates all of the appropriated control limits, based on the current data. The example below is extracted from the WERulesVariableControlChart.XBarRCharts example program.

If you want to include the WE Trending (Supplemental) rules, in addition to the regular WE Runtime rules, call UseWERuntimeAndSupplementalRules in place of UseWERuntimeRules.

```
primaryChart.useWERuntimeRules();
// IMPORTANT TO ADD THIS LINE to process alarm event alarmEventChanged below
// This triggers an alarm based on the state of the alarm, not a transition.
chartdata.addAlarmStateEventListener(this);
// don't generate alarms in initial data simulation
chartdata.setAlarmStateEventEnable(false);


.
.
.

// Must have data loaded before any of the Auto.. methods are called
simulateData(150, 30);


// Calculate the SPC control limits for both graphs of the current SPC chart (X-
Bar R)
this.autoCalculateControlLimits();


// start checking alarm limits now
```

```
chartdata.setAlarmStateEventEnable(true);
simulateData(150, 32);
```

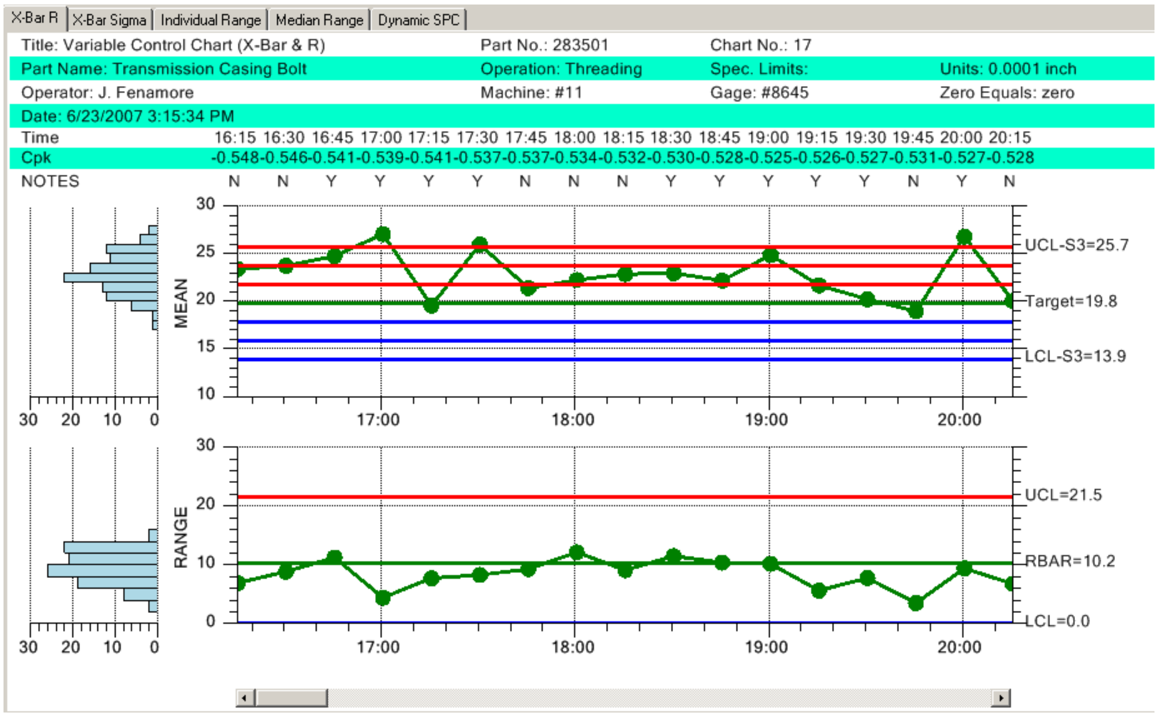
If you have setup a method for processing alarm events, the software will call the classes alarm processing method, where you can take appropriate action. If a time interval has multiple alarms, i.e. more than one of the four WR Runtime rules are broken, only the one with the lowest WE rule number is vectored to the alarm event processing routine.

```
public class XBarRChart extends SPCTimeVariableControlChart implements
SPCAlarmEventListener {
 .
.
.
public void initializeChart()
{
  .
  .
  .

  primaryChart.useWERuntimeRules();
    // IMPORTANT TO ADD THIS LINE to process alarm event alarmEventChanged below
- MAY NOT BE IN MANUAL
    // This triggers an alarm based on the state of the alarm, not a transition.
//   Adjacent alarm conditions will continue to
    // trigger the alarm even.
    chartdata.addAlarmStateEventListener(this);
    // don't generate alarms in initial data simulation
    chartdata.setAlarmStateEventEnable(false);
  .
  .
  .
// Must have data loaded before any of the Auto.. methods are called
  simulateData(150, 30);
// Calculate the SPC control limits for both graphs
  this.autoCalculateControlLimits();

// start checking alarm limits now
  chartdata.setAlarmStateEventEnable(true);

  simulateData(150, 32);
  .
  .
  .
}

public void alarmEventChanged(SPCControlChartData sender, SPCControlLimitAlarmArgs
e)
{
    SPCControlLimitRecord alarm = e.getEventAlarm();
    double alarmlimitvalue = alarm.getControlLimitValue();
    String alarmlimitvaluestring = Double.toString(alarmlimitvalue);

    SPCControlChartData spcData = alarm.getSPCProcessVar();
    SPCCalculatedValueRecord spcSource = e.getSPCSource();
    String calculatedvaluestring =
        Double.toString(spcSource.getCalculatedValue());

    String message = alarm.getAlarmMessage();
    GregorianCalendar timestamp = spcData.getTimeStamp();
    String timestampstring = ChartCalendar.toString(timestamp);
    String notesstring = "\n" + timestampstring + " " + message + "=" + "\n" +

    alarmlimitvaluestring + " Current Value" + "=" + calculatedvaluestring;

    if (alarm.getAlarmState())
        this.getChartData().appendNotesString(notesstring, true);

}
```

If you want multiple alarms for a time interval vectored to the alarm processing routine (i.e. it is possible that a time period has WE1, WE2, WE3 and WE4 alarms), set the SPCControlChartData property to SPCControlChartData .REPORT_ALL_ALARMS.

```
this.getChartData().setAlarmReportMode( SPCControlChartData.REPORT_ALL_ALARMS);
```

The resulting X-Bar R SPC Chart with WE Runtime Rules looks something like this. In this example, the WR Rules violations are processed by the **SPCControlLimitAlarm** method, where the alarm condition is added to the Notes record for the appropriate sample interval. The Y in the Notes line indicates that an alarm record has been saved for that time interval, and you can click on the Y to see the note describing the alarm condition.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch
Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero
Date: 6/23/2007 3:15:34 PM

Time	16:15	16:30	16:45	17:00	17:15	17:30	17:45	18:00	18:15	18:30	18:45	19:00	19:15	19:30	19:45	20:00	20:15
Cpk	-0.548	-0.546	-0.541	-0.539	-0.541	-0.537	-0.537	-0.534	-0.532	-0.530	-0.528	-0.525	-0.526	-0.527	-0.531	-0.527	-0.528
NOTES	N	N	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	Y	N	Y	N

Specification Limits

Specification limits are not to be confused with the SPC Control Limits discussed in the previous sections. Specification limits are imposed externally and are not calculated based on the manufacturing process under control. They represent the maximum deviation allowable for the process variable being measured. They are calculated based on input from customers and/or engineering. Usually specification limits are going to be wider than the SPC 3-sigma limits, because you want the SPC control limits to trip before you get to the specification limits. The SPC control limits give you advance notice that the process is going south before you start rejecting parts based on specification limits. You can display specification limits in the same chart as SPC control limits. Use the addSpecLimit method of the PrimaryChart or SecondaryChart.

```
this.getPrimaryChart().addSpecLimit(SPCChartObjects.SPC_LOWER_SPEC_LIMIT, 18.3, "L
SPEC", new ChartAttribute(ChartColors.GREEN, 3.0));
this.getPrimaryChart().addSpecLimit(SPCChartObjects.SPC_UPPER_SPEC_LIMIT, 39.1, "H
SPEC", new ChartAttribute(ChartColors.YELLOW, 3.0));
```

Chart Y-Scale

You can set the minimum and maximum values of the two charts y-scales manually using the **getPrimaryChart().setMinY**, **getPrimaryChart().setMaxY**, **getSecondaryChart().setMinY** and **getSecondaryChart().setMaxY** properties.

```
// Set initial scale of the y-axis of the mean chart
// If you are calling AutoScalePrimaryChartYRange this isn't really needed
    this.getPrimaryChart().setMinY( 0);
    this.getPrimaryChart().setMaxY( 40)

// Set initial scale of the y-axis of the range chart
// If you are calling AutoScaleSecondaryChartYRange this isn't really needed
    this.getSecondaryChart().setMinY( 0);
    this.getSecondaryChart().setMaxY( 40);
```

It is easiest to just call the auto-scale routines after the chart has been initialized with data, and any control limits calculated.

```
// Must have data loaded before any of the Auto.. methods are called
    simulateData();

// Calculate the SPC control limits for both graphs of the current SPC chart
    this.autoCalculateControlLimits ();

// Scale the y-axis of the X-Bar chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
// Scale the y-axis of the Range chart to display all data and control limits
    this.autoScaleSecondaryChartYRange();
```

Once all of the graph parameters are set, call the method **rebuildChartUsingCurrentData** .

```
// Rebuild the chart using the current data and settings
this.rebuildChartUsingCurrentData ();
```

If, at any future time you change any of the chart properties, you will need to call **rebuildChartUsingCurrentData** to force a rebuild of the chart, taking into account the current properties. **rebuildChartUsingCurrentData** also invalidates the chart and forces a redraw. Our examples that update dynamically demonstrate this technique. The chart is setup with some initial settings and data values. As data is added in real-time to the graph, the chart SPC limits, and y-scales are constantly recalculated to take into account new data values. The following code is extracted from the TimeVariableControlCharts.DynamicXBarRChart example.

```
private void timer1_Tick()
{
    GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();

    // Use the ChartData sample simulator to make an array of sample data
    DoubleArray samples = this.getChartData().simulateMeasurementRecord(30);
    // Add the new sample subgroup to the chart
    this.getChartData().addNewSampleRecord(timestamp, samples);

// Calculate the SPC control limits for the X-Bar part of the current SPC chart
    this.autoCalculateControlLimits();
    // Scale the y-axis of the X-Bar chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    this.autoScaleSecondaryChartYRange();
    // Rebuild and draw the chart using the current data and settings
    this.rebuildChartUsingCurrentData();
    // Simulate timeincrementminutes  minute passing
    startTime.add(ChartObj.MINUTE, timeincrementminutes);
}
```

Updating Chart Data

The real-time example above demonstrates how the SPC chart data is updated, using the **getChartData().addNewSampleRecord** method. In this case, the chart data updates with each timer tick event, though it could just as easily be any other type of event. If you have already collected all of your data and just want to plot it all at once, use a simple loop like most of our examples do to update the data.

```
private void simulateData()
{   for (int i=0; i < 200; i++)
    {
        GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
        // Use the ChartData sample simulator to make an array of sample data

    DoubleArray samples = this.getChartData().simulateMeasurementRecord(30, 10);
        // Add the new sample subgroup to the chart
        this.getChartData().addNewSampleRecord(timestamp, samples);
        // increment simulated time by timeincrementminutes minutes
        startTime.add(ChartObj.MINUTE, timeincrementminutes);
    }
}
```

In this example the sample data and the time stamp for each sample record is simulated. In your application, you will probably be reading the sample record values from some sort of database or file, along with the actual time stamp for that data.

If you want to append a text note to a sample record, use one of the **getChartData().addNewSampleRecord** overrides that have a *notes* parameter.

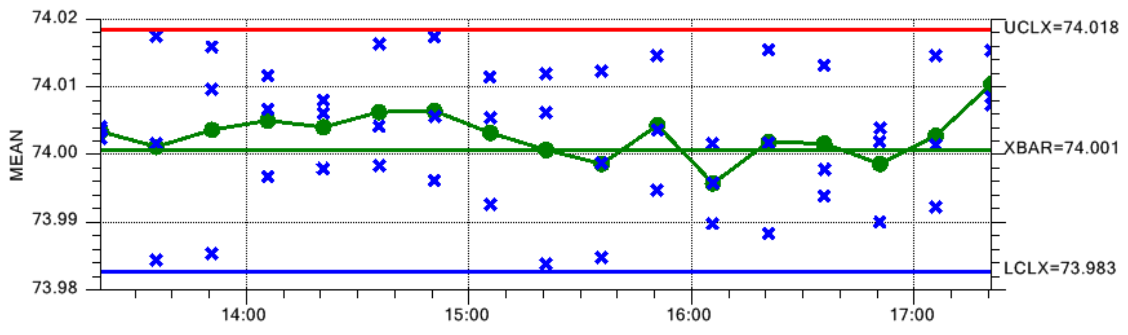
```
private void simulateData()
{  String notesstring = "";
    for (int i=0; i < 200; i++)
    {
    GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
    // Use the ChartData sample simulator to make an array of sample data

    DoubleArray samples = this.getChartData().simulateMeasurementRecord(30, 10);
    double r = ChartSupport.getRandomDouble();
    if (r < 0.1) // make a note on every tenth item, on average
        notesstring = "Note for sample subgroup #" + Integer.toString(i) + ".
Lathe cutting tool broke. Replaced with new, Aeon cutting tool.";
    else
        notesstring = "";
    // Add the new sample subgroup to the chart
    this.getChartData().addNewSampleRecord(timestamp, samples, notesstring);
    // increment simulated time by timeincrementminutes minutes
    startTime.add(ChartObj.MINUTE, timeincrementminutes);
    }
}
```

There are situations where you might want to add, change, modify, or append a note for a sample subgroup after the **addNewSampleRecord** method has already been called for the sample subgroup. This can happen if the **addNewSampleRecord** method call generates an alarm event. In the alarm event processing routine, you can add code that adds a special note to the sample subgroup that generated the alarm. Use the **ChartData.setNotesString** or **ChartData.appendNotesString** methods to add notes to the current record, separate from the **addNewSampleRecord** method.

Extracted from the VariableControlCharts.DynamicXBarRChart example program.

```
public void alarmEventChanged(SPCControlChartData sender,
    SPCControlLimitAlarmArgs e)
{
    SPCControlLimitRecord alarm = e.getEventAlarm();
    double alarmlimitvalue = alarm.getControlLimitValue();
    String alarmlimitvaluestring = Double.toString(alarmlimitvalue);
    SPCControlChartData spcData = alarm.getSPCProcessVar();
    SPCCalculatedValueRecord spcSource = e.getSPCSource();
    String calculatedvaluestring = ouble.toString(spcSource.getCalculatedValue());

    String message = alarm.getAlarmMessage();
    GregorianCalendar timestamp = spcData.getTimeStamp();
    String timestampstring = timestamp.toString();
    String notesstring = "\n" + timestampstring + " " + message + "=" + "\n" +

        alarmlimitvaluestring + " Current Value" + "=" + calculatedvaluestring;

    if (alarm.getAlarmState())
            this.getChartData().appendNotesString(notesstring, true);

}
```

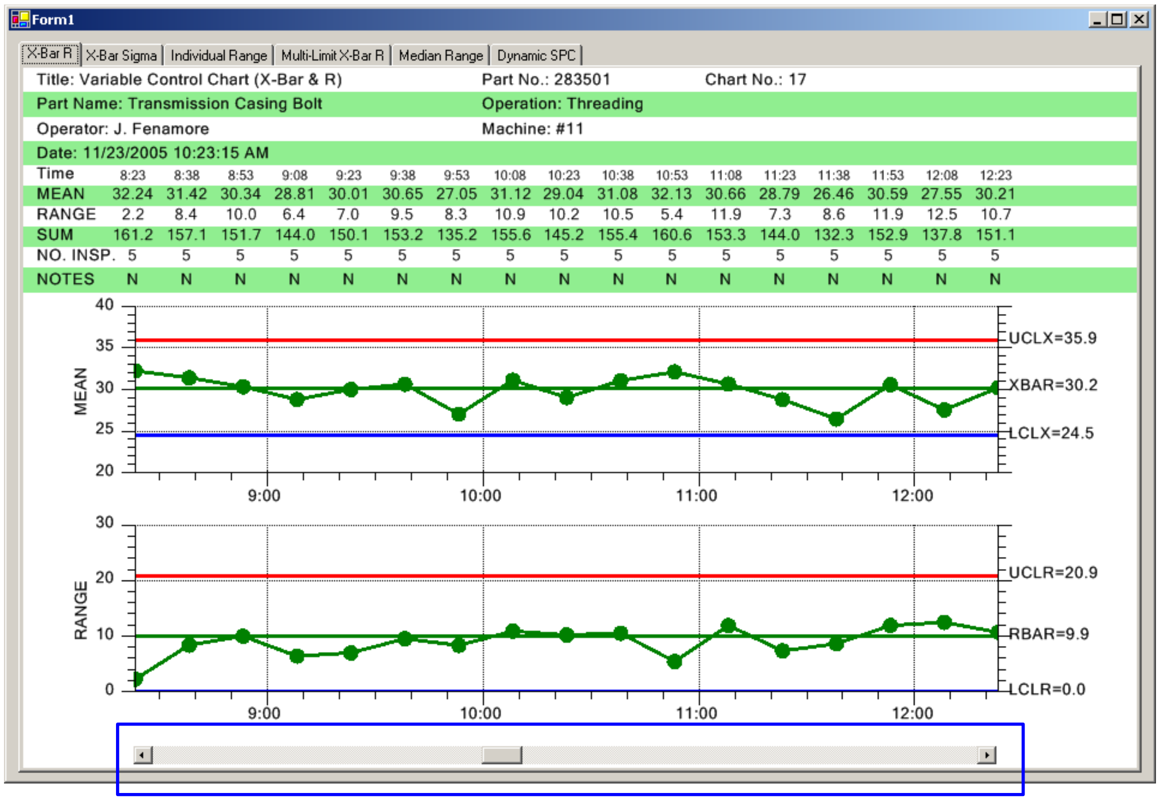
Scatter Plots of the Actual Sampled Data



If you want the actual sample data plotted along with the mean or median of the sample data, set the **getPrimaryChart().setPlotMeasurementValues** to true.

```
// Plot individual sampled values as a scatter plot
this.getPrimaryChart().setPlotMeasurementValues( true);
```

Enable the Chart ScrollBar



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 11/23/2005 10:23:15 AM

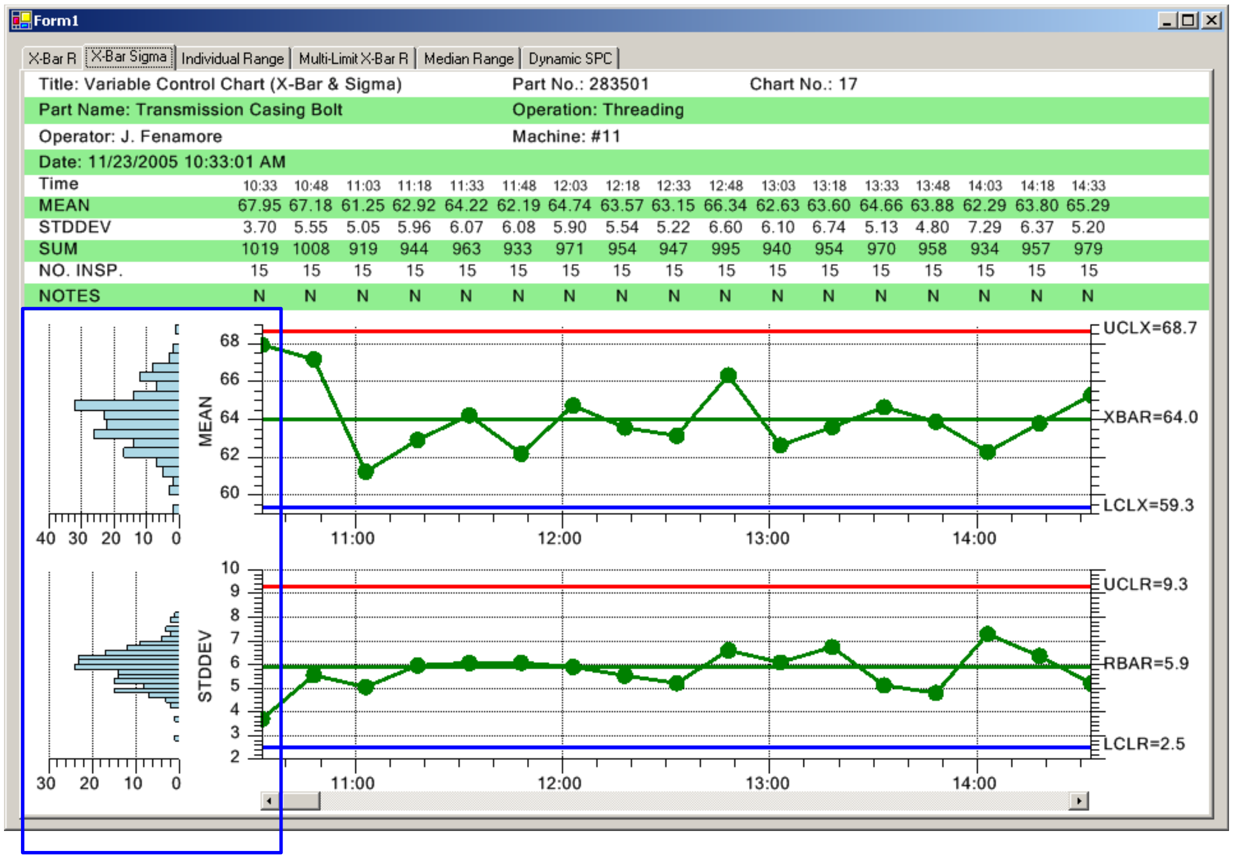
Time	8:23	8:38	8:53	9:08	9:23	9:38	9:53	10:08	10:23	10:38	10:53	11:08	11:23	11:38	11:53	12:08	12:23
MEAN	32.24	31.42	30.34	28.81	30.01	30.65	27.05	31.12	29.04	31.08	32.13	30.66	28.79	26.46	30.59	27.55	30.21
RANGE	2.2	8.4	10.0	6.4	7.0	9.5	8.3	10.9	10.2	10.5	5.4	11.9	7.3	8.6	11.9	12.5	10.7
SUM	161.2	157.1	151.7	144.0	150.1	153.2	135.2	155.6	145.2	155.4	160.6	153.3	144.0	132.3	152.9	137.8	151.1
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Scrollbar

Set the **EnableScrollBar** property true to enable the chart scrollbar. You will then be able to window in on 8-20 sample subgroups at a time, from a much larger collection of measurement data representing hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.

```
// enable scroll bar
this.setEnableScrollBar(true);
```

SPC Chart Histograms



Title: Variable Control Chart (X-Bar & Sigma) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Fenamore Machine: #11

Date: 11/23/2005 10:33:01 AM

Time	10:33	10:48	11:03	11:18	11:33	11:48	12:03	12:18	12:33	12:48	13:03	13:18	13:33	13:48	14:03	14:18	14:33
MEAN	67.95	67.18	61.25	62.92	64.22	62.19	64.74	63.57	63.15	66.34	62.63	63.60	64.66	63.88	62.29	63.80	65.29
STDDEV	3.70	5.55	5.05	5.96	6.07	6.08	5.90	5.54	5.22	6.60	6.10	6.74	5.13	4.80	7.29	6.37	5.20
SUM	1019	1008	919	944	963	933	971	954	947	995	940	954	970	958	934	957	979
NO. INSP.	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Frequency Histograms

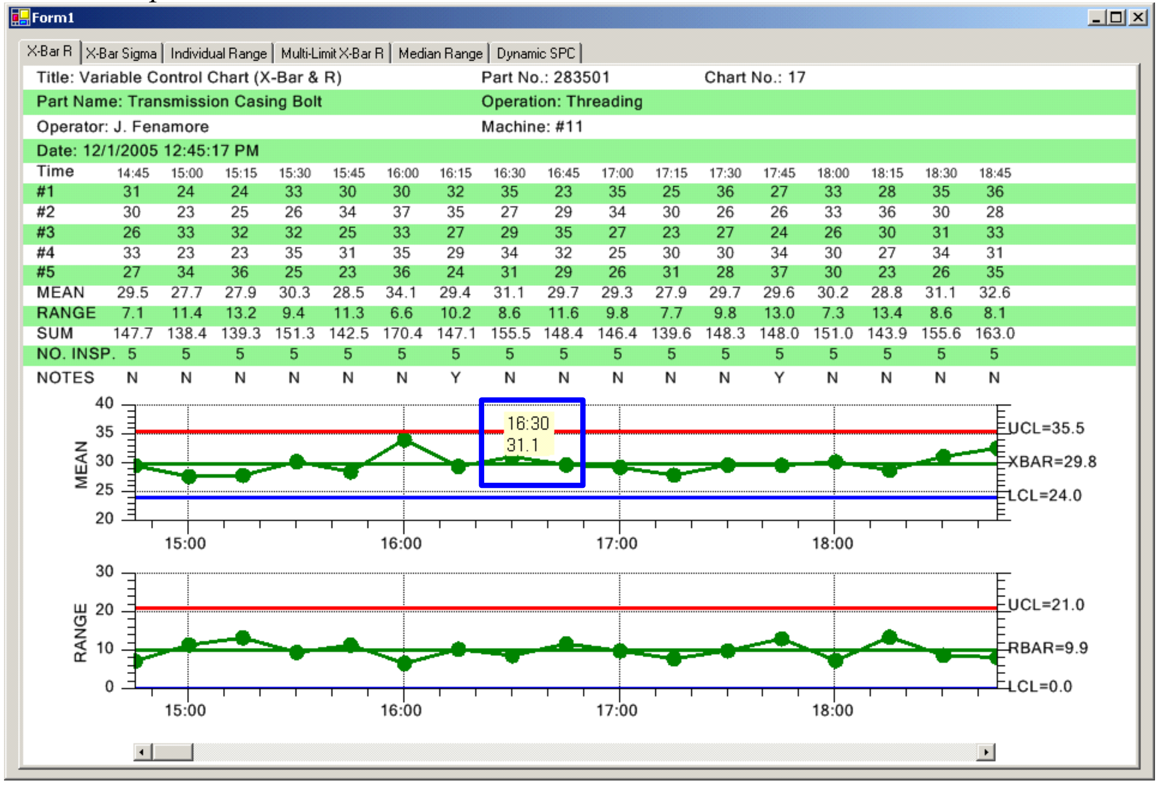
Viewing frequency histograms of both the variation in the primary variable (Mean, Median, count, or actual value), and the secondary variable (Range, Sigma or Moving Range) side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process. You can turn on integrated frequency histograms for either chart using the **getPrimaryChart().DisplayFrequencyHistogram** and **getSecondaryChart().DisplayFrequencyHistogram** properties of the chart.

```
//  frequency histogram for both charts
this.getPrimaryChart().setDisplayFrequencyHistogram( true);
this.getSecondaryChart().setDisplayFrequencyHistogram( true);
```

SPC Chart Data and Notes Tooltips

You can invoke two types of tooltips using the mouse. The first is a data tooltip. When you hold the mouse button down over one of the data points, in the primary or secondary chart, the x and y values for that data point display in a popup tooltip.

Data Tooltip



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 12/1/2005 12:45:17 PM

Time	14:45	15:00	15:15	15:30	15:45	16:00	16:15	16:30	16:45	17:00	17:15	17:30	17:45	18:00	18:15	18:30	18:45
#1	31	24	24	33	30	30	32	35	23	35	25	36	27	33	28	35	36
#2	30	23	25	26	34	37	35	27	29	34	30	26	26	33	36	30	28
#3	26	33	32	32	25	33	27	29	35	27	23	27	24	26	30	31	33
#4	33	23	23	35	31	35	29	34	32	25	30	30	34	30	27	34	31
#5	27	34	36	25	23	36	24	31	29	26	31	28	37	30	23	26	35
MEAN	29.5	27.7	27.9	30.3	28.5	34.1	29.4	31.1	29.7	29.3	27.9	29.7	29.6	30.2	28.8	31.1	32.6
RANGE	7.1	11.4	13.2	9.4	11.3	6.6	10.2	8.6	11.6	9.8	7.7	9.8	13.0	7.3	13.4	8.6	8.1
SUM	147.7	138.4	139.3	151.3	142.5	170.4	147.1	155.5	148.4	146.4	139.6	148.3	148.0	151.0	143.9	155.6	163.0
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NOTES	N	N	N	N	N	N	Y	N	N	N	N	N	Y	N	N	N	N

In the default mode, the data tooltip displays the x,y value of the data point nearest the mouse click. If the x-axis is a time axis then the x-value is displayed as a time stamp; otherwise, it is displayed as a simple numeric value, as is the y-value. You can optionally display subgroup informaiton (sample values, calculated values, process capability values and notes) in the data tooltip window, under the x,y value, using enable flags in the primary charts tooltip property.

Extracted from the TimeVariableControlCharts.XBarRChart example.

```
SPCChartObjects primaryChart = this.getPrimaryChart();


primaryChart.getDatatooltip().setEnableCategoryValues(true);
primaryChart.getDatatooltip().setEnableCalculatedValues(true);
primaryChart.getDatatooltip().setEnableProcessCapabilityValues(true);
primaryChart.getDatatooltip().setEnableCategoryValues(true);
```

where

The following methods data tooltip methods enable sections of the chart header and table:

setEnableCategoryValues
setEnableProcessCapabilityValues
setEnableCalculatedValues
setEnableNotesStrings

Display the category (subgroup sample values) in the data tooltip.

```
primaryChart.getDatatooltip().setEnableCategoryValues(true);
```

Display the calculated values used in the chart (Mean, range and sum for an Mean-Range chart).

```
primaryChart.getDatatooltip().setEnableCalculatedValues(true);
```

Display the process capability (Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu and Ppk) statistics currently being calculated for the chart.

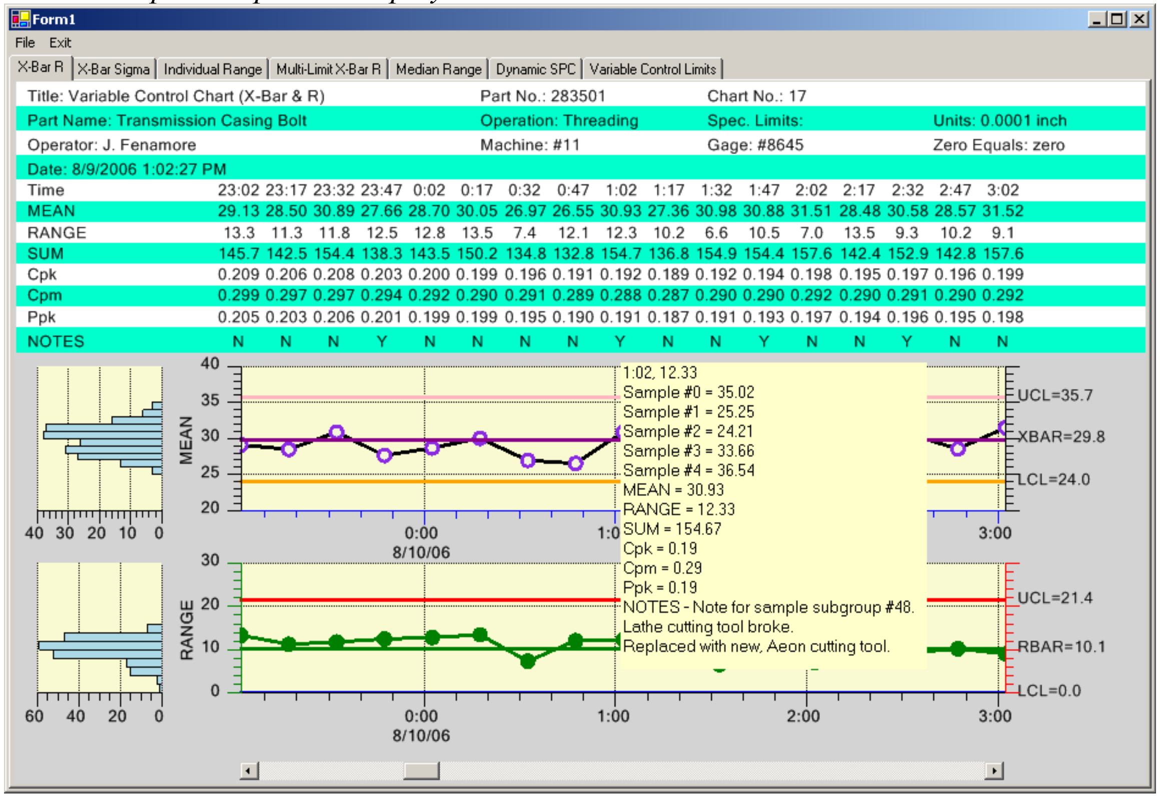
```
primaryChart.getDatatooltip().setEnableProcessCapabilityValues(true);
```

Display the current notes string for the sample subgroup.

```
primaryChart.getDatatooltip().setEnableCategoryValues(true);
```

The variable control chart below displays a tooltip with all of the enable options above set true.

Data Tooltip with optional display items



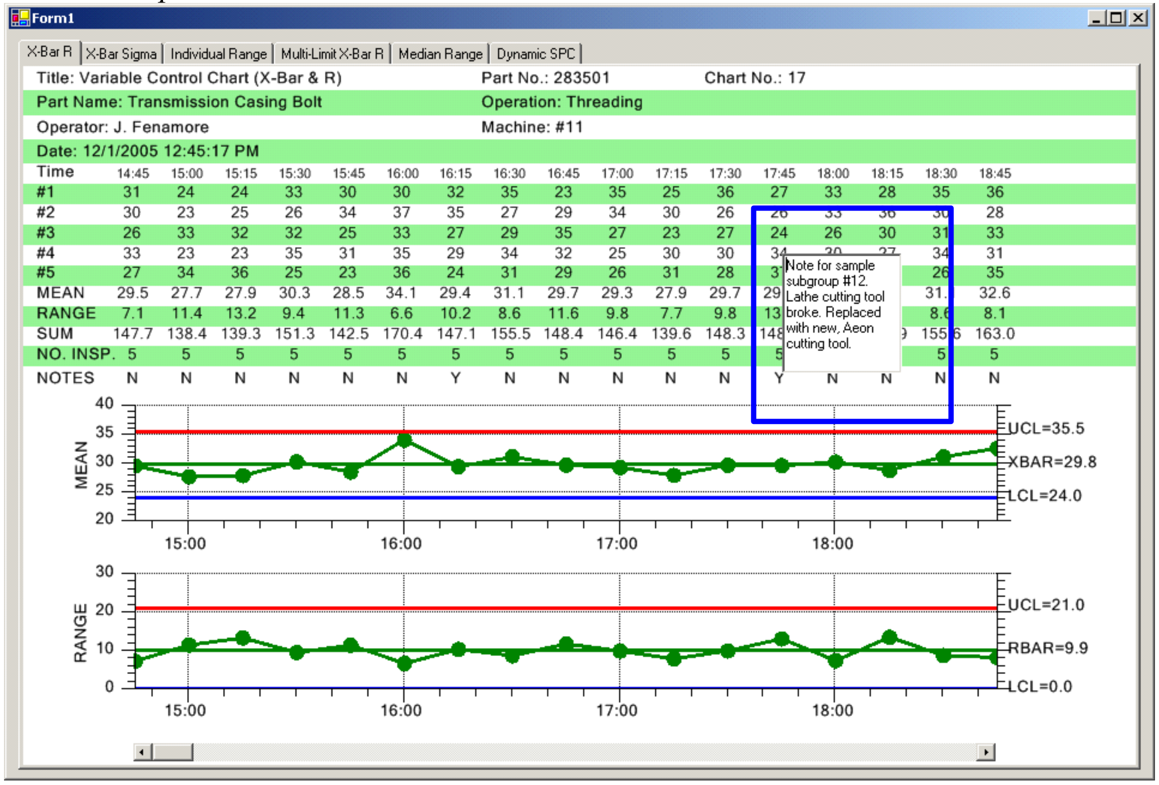
Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation: Threading	Spec. Limits:	Units: 0.0001 inch
Operator: J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 8/9/2006 1:02:27 PM			

Time	23:02	23:17	23:32	23:47	0:02	0:17	0:32	0:47	1:02	1:17	1:32	1:47	2:02	2:17	2:32	2:47	3:02
MEAN	29.13	28.50	30.89	27.66	28.70	30.05	26.97	26.55	30.93	27.36	30.98	30.88	31.51	28.48	30.58	28.57	31.52
RANGE	13.3	11.3	11.8	12.5	12.8	13.5	7.4	12.1	12.3	10.2	6.6	10.5	7.0	13.5	9.3	10.2	9.1
SUM	145.7	142.5	154.4	138.3	143.5	150.2	134.8	132.8	154.7	136.8	154.9	154.4	157.6	142.4	152.9	142.8	157.6
Cpk	0.209	0.206	0.208	0.203	0.200	0.199	0.196	0.191	0.192	0.189	0.192	0.194	0.198	0.195	0.197	0.196	0.199
Cpm	0.299	0.297	0.297	0.294	0.292	0.290	0.291	0.289	0.288	0.287	0.290	0.290	0.292	0.290	0.291	0.290	0.292
Ppk	0.205	0.203	0.206	0.201	0.199	0.199	0.195	0.190	0.191	0.187	0.191	0.193	0.197	0.194	0.196	0.195	0.198
NOTES	N	N	N	Y	N	N	N	N	Y	N	N	Y	N	N	Y	N	N

1:02, 12.33
Sample #0 = 35.02
Sample #1 = 25.25
Sample #2 = 24.21
Sample #3 = 33.66
Sample #4 = 36.54
MEAN = 30.93
RANGE = 12.33
SUM = 154.67
Cpk = 0.19
Cpm = 0.29
Ppk = 0.19
NOTES - Note for sample subgroup #48.
Lathe cutting tool broke.
Replaced with new, Aeon cutting tool.

If you are displaying the Notes line in the table portion of the chart, the Notes entry for a sample subgroup will display "Y" if a note was recorded for that sample subgroup, or "N" if no note was recorded. Notes are recorded using one of the **getChartData().addNewSampleRecord** overrides that include a notes parameter. See the section *Updating Chart Data*. If you click on a "Y" in the Notes row for a sample subgroup, the complete text of the note for that sample subgroup will display in a JTextArea, immediately above the "Y". You can actually edit the notes in the JTextArea.

Notes Tooltip



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Fenamore Machine: #11

Date: 12/1/2005 12:45:17 PM

Time	14:45	15:00	15:15	15:30	15:45	16:00	16:15	16:30	16:45	17:00	17:15	17:30	17:45	18:00	18:15	18:30	18:45
#1	31	24	24	33	30	30	32	35	23	35	25	36	27	33	28	35	36
#2	30	23	25	26	34	37	35	27	29	34	30	26	26	33	36	30	28
#3	26	33	32	32	25	33	27	29	35	27	23	27	24	26	30	31	33
#4	33	23	23	35	31	35	29	34	32	25	30	30	34	30	27	34	31
#5	27	34	36	25	23	36	24	31	29	26	31	28	31	[illegible]	[illegible]	26	35
MEAN	29.5	27.7	27.9	30.3	28.5	34.1	29.4	31.1	29.7	29.3	27.9	29.7	29	[illegible]	[illegible]	31.	32.6
RANGE	7.1	11.4	13.2	9.4	11.3	6.6	10.2	8.6	11.6	9.8	7.7	9.8	13	[illegible]	[illegible]	8.6	8.1
SUM	147.7	138.4	139.3	151.3	142.5	170.4	147.1	155.5	148.4	146.4	139.6	148.3	148	[illegible]	9	155.6	163.0
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	[illegible]	[illegible]	5	5
NOTES	N	N	N	N	N	N	Y	N	N	N	N	N	Y	N	N	N	N

Note for sample subgroup #12. Lathe cutting tool broke. Replaced with new, Aeon cutting tool.

```
private void simulateData()
{   String notesstring = "";
    for (int i=0; i < 200; i++)
    {
        .
        .
        .
        this.getChartData().addNewSampleRecord (timestamp, samples, notesstring);
        // increment simulated time by timeincrementminutes minutes
        startTime.add(ChartObj.MINUTE, timeincrementminutes);
    }
}
```

Both kinds of tooltips are on by default. Turn the tooltips on or off in the program using the **EnableDataToolTip** and **EnableNotesToolTip** flags.

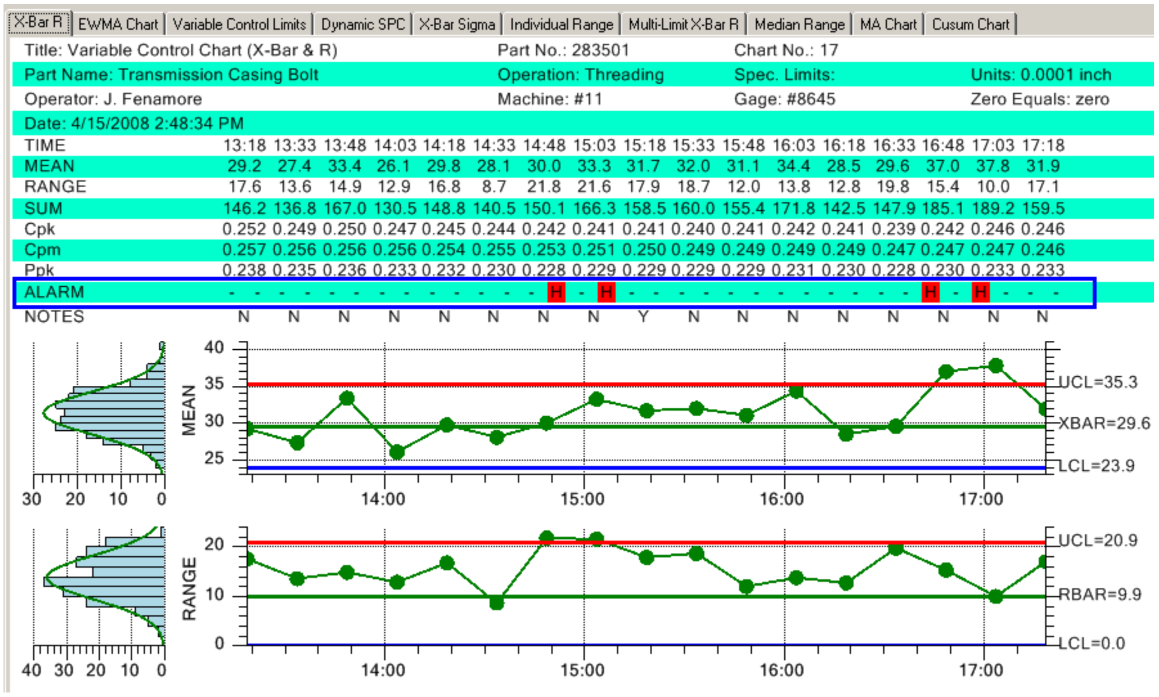
```
// Enable data and notes tooltips
this.setEnableDataToolTip(true);
this.setEnableNotesToolTip(true);
```

The notes tooltip has an additional option. In order to make the notes tooltip “editable”, the tooltip, which is Java **JTextArea**, displays on the first click, and goes away on the second click. You can click inside the **JTextArea** and not worry the tooltip suddenly disappearing. The notes tooltip works this way by default. If you wish to explicitly set it, or change it so that the tooltip only displays while the mouse button is held down, as the data tooltips do, set the **getChartData().getNotesToolTips().ToolTipMode** property to **NotesToolTip.MOUSEDOWN_TOOLTIP**, as in the example below.

```
// Enable data and notes tooltips

this.setEnableDataToolTip = true;

this.setEnableNotesToolTip = true;


this.getChartData().getNotesToolTips.setButtonMask(InputEvent.BUTTON1_MASK);

// default is MOUSETOGGLE_TOOLTIP

this.getChartData().getNotesToolTips().setToolTipMode( NotesToolTi
p.MOUSEDOWN_TOOLTIP);
```

Enable Alarm Highlighting

EnableAlarmStatusValues



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch
Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero
Date: 4/15/2008 2:48:34 PM

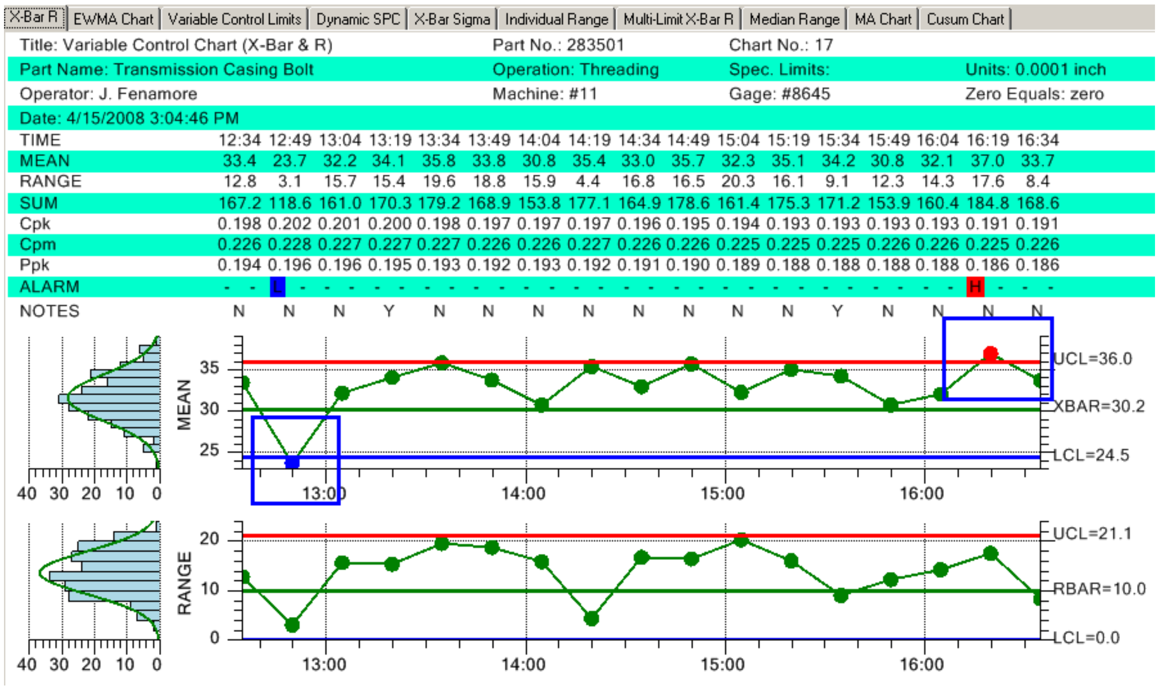
TIME	13:18	13:33	13:48	14:03	14:18	14:33	14:48	15:03	15:18	15:33	15:48	16:03	16:18	16:33	16:48	17:03	17:18
MEAN	29.2	27.4	33.4	26.1	29.8	28.1	30.0	33.3	31.7	32.0	31.1	34.4	28.5	29.6	37.0	37.8	31.9
RANGE	17.6	13.6	14.9	12.9	16.8	8.7	21.8	21.6	17.9	18.7	12.0	13.8	12.8	19.8	15.4	10.0	17.1
SUM	146.2	136.8	167.0	130.5	148.8	140.5	150.1	166.3	158.5	160.0	155.4	171.8	142.5	147.9	185.1	189.2	159.5
Cpk	0.252	0.249	0.250	0.247	0.245	0.244	0.242	0.241	0.241	0.240	0.241	0.242	0.241	0.239	0.242	0.246	0.246
Cpm	0.257	0.256	0.256	0.256	0.254	0.255	0.253	0.251	0.250	0.249	0.249	0.249	0.249	0.247	0.247	0.247	0.246
Ppk	0.238	0.235	0.236	0.233	0.232	0.230	0.228	0.229	0.229	0.229	0.229	0.231	0.230	0.228	0.230	0.233	0.233
ALARM	- -	- -	- -	- -	- -	- -	- H	- H	- -	- -	- -	- -	- -	- -	H -	H -	- -
NOTES	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	N	N	N

There are several alarm highlighting options you can turn on and off. The alarm status line above is turned on/off using the EnableAlarmStatusValues property. We have set it on by default, so you will have to turn it off if you don’t want it. Each sample interval has two small boxes that are labeled using one of several different characters, listed below. The most common are an “H” signifying a high alarm, a “L” signifying a low alarm, and a “-“ signifying that there is no alarm. When specialized control rules are implemented, either using the named rules discussed in Chapter 8, or custom rules involving trending, oscillation, or stratification, a “T”, “O” or “S” may also appear.

“-”	No alarm condition
“H”	High - Measured value is above a high limit
“L”	Low - Measured value falls below a low limit
“T”	Trending - Measured value is trending up (or down).
“O”	Oscillation - Measured value is oscillating (alternating) up and down.
“S”	Stratification - Measured value is stuck in a narrow band.

```
// Alarm status line
this.setEnableAlarmStatusValues( false);
```

ChartAlarmEmphasisMode



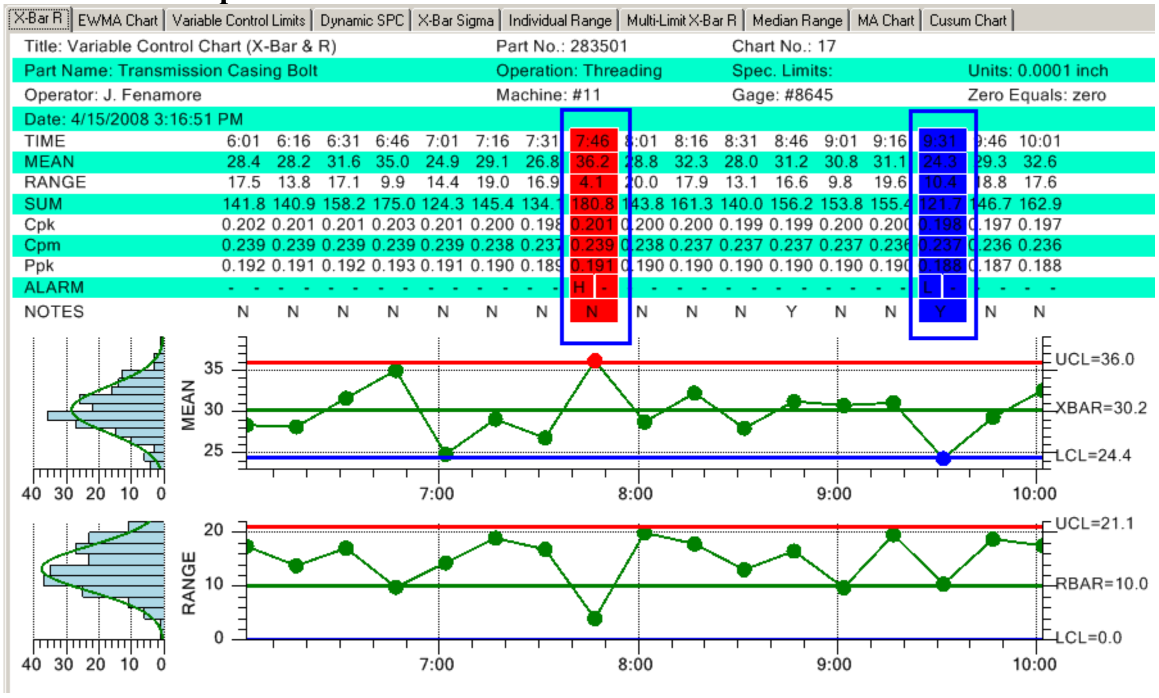
Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch
Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero
Date: 4/15/2008 3:04:46 PM

TIME	12:34	12:49	13:04	13:19	13:34	13:49	14:04	14:19	14:34	14:49	15:04	15:19	15:34	15:49	16:04	16:19	16:34
MEAN	33.4	23.7	32.2	34.1	35.8	33.8	30.8	35.4	33.0	35.7	32.3	35.1	34.2	30.8	32.1	37.0	33.7
RANGE	12.8	3.1	15.7	15.4	19.6	18.8	15.9	4.4	16.8	16.5	20.3	16.1	9.1	12.3	14.3	17.6	8.4
SUM	167.2	118.6	161.0	170.3	179.2	168.9	153.8	177.1	164.9	178.6	161.4	175.3	171.2	153.9	160.4	184.8	168.6
Cpk	0.198	0.202	0.201	0.200	0.198	0.197	0.197	0.197	0.196	0.195	0.194	0.193	0.193	0.193	0.193	0.191	0.191
Cpm	0.226	0.228	0.227	0.227	0.227	0.226	0.226	0.227	0.226	0.226	0.225	0.225	0.225	0.226	0.226	0.225	0.226
Ppk	0.194	0.196	0.196	0.195	0.193	0.192	0.193	0.192	0.191	0.190	0.189	0.188	0.188	0.188	0.188	0.186	0.186
ALARM	- -	L -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	H -	- -
NOTES	N	N	N	Y	N	N	N	N	N	N	N	N	Y	N	N	N	N

```
// Chart alarm emphasis mode
this.setChartAlarmEmphasisMode( SPCChartBase.ALARM_HIGHLIGHT_SYMBOL);
```

The scatter plot symbol used to plot a data point in the primary and secondary charts is normally a fixed color circle. If you turn on the alarm highlighting for chart symbols the symbol color for a sample interval that is in an alarm condition will change to reflect the color of the associated alarm line. In the example above, a low alarm (blue circle) occurs at the beginning of the chart and a high alarm (red circle) occurs at the end of the chart. Alarm symbol highlighting is turned on by default. To turn it off use the SPCChartBase.ALARM_NO_HIGHLIGHT_SYMBOL constants.

TableAlarmEmphasisMode -



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch
Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero
Date: 4/15/2008 3:16:51 PM

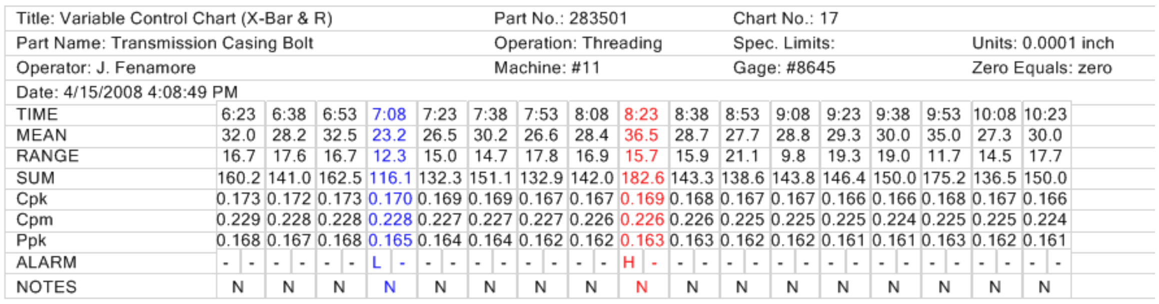
TIME	6:01	6:16	6:31	6:46	7:01	7:16	7:31	7:46	8:01	8:16	8:31	8:46	9:01	9:16	9:31	9:46	10:01
MEAN	28.4	28.2	31.6	35.0	24.9	29.1	26.8	36.2	28.8	32.3	28.0	31.2	30.8	31.1	24.3	29.3	32.6
RANGE	17.5	13.8	17.1	9.9	14.4	19.0	16.9	4.1	20.0	17.9	13.1	16.6	9.8	19.6	10.4	18.8	17.6
SUM	141.8	140.9	158.2	175.0	124.3	145.4	134.1	180.8	143.8	161.3	140.0	156.2	153.8	155.4	121.7	146.7	162.9
Cpk	0.202	0.201	0.201	0.203	0.201	0.200	0.198	0.201	0.200	0.200	0.199	0.199	0.200	0.200	0.198	0.197	0.197
Cpm	0.239	0.239	0.239	0.239	0.239	0.238	0.237	0.239	0.238	0.237	0.237	0.237	0.237	0.236	0.237	0.236	0.236
Ppk	0.192	0.191	0.192	0.193	0.191	0.190	0.189	0.191	0.190	0.190	0.190	0.190	0.190	0.190	0.188	0.187	0.188
ALARM	- -	- -	- -	- -	- -	- -	- -	H -	- -	- -	- -	- -	- -	- -	L -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N

```
// Table alarm emphasis mode
this.setTableAlarmEmphasisMode (SPCChartBase.ALARM_HIGHLIGHT_BAR);
```

The entire column of the data table can be highlighted when an alarm occurs. There are four modes associated with this property:

ALARM_HIGHLIGHT_NONE	No alarm highlight
ALARM_HIGHLIGHT_TEXT	Text alarm highlight
ALARM_HIGHLIGHT_OUTLINE	Outline alarm highlight
ALARM_HIGHLIGHT_BAR	Bar alarm highlight

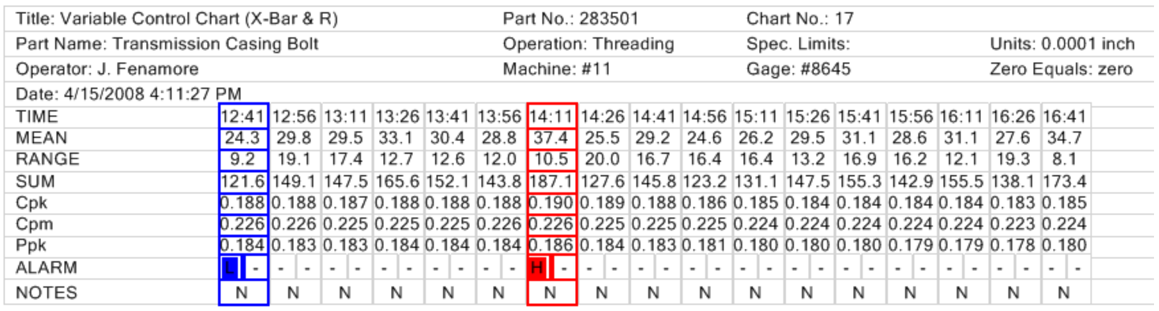
The example above uses the ALARM_HIGHLIGHT_BAR mode.



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation: Threading	Spec. Limits:	Units: 0.0001 inch
Operator: J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 4/15/2008 4:08:49 PM			

TIME	6:23	6:38	6:53	7:08	7:23	7:38	7:53	8:08	8:23	8:38	8:53	9:08	9:23	9:38	9:53	10:08	10:23
MEAN	32.0	28.2	32.5	23.2	26.5	30.2	26.6	28.4	36.5	28.7	27.7	28.8	29.3	30.0	35.0	27.3	30.0
RANGE	16.7	17.6	16.7	12.3	15.0	14.7	17.8	16.9	15.7	15.9	21.1	9.8	19.3	19.0	11.7	14.5	17.7
SUM	160.2	141.0	162.5	116.1	132.3	151.1	132.9	142.0	182.6	143.3	138.6	143.8	146.4	150.0	175.2	136.5	150.0
Cpk	0.173	0.172	0.173	0.170	0.169	0.169	0.167	0.167	0.169	0.168	0.167	0.167	0.166	0.166	0.168	0.167	0.166
Cpm	0.229	0.228	0.228	0.228	0.227	0.227	0.227	0.226	0.226	0.226	0.225	0.225	0.225	0.224	0.225	0.225	0.224
Ppk	0.168	0.167	0.168	0.165	0.164	0.164	0.162	0.162	0.163	0.163	0.162	0.162	0.161	0.161	0.163	0.162	0.161
ALARM	- -	- -	- -	L -	- -	- -	- -	- -	H -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_TEXT mode



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation: Threading	Spec. Limits:	Units: 0.0001 inch
Operator: J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 4/15/2008 4:11:27 PM			

TIME	12:41	12:56	13:11	13:26	13:41	13:56	14:11	14:26	14:41	14:56	15:11	15:26	15:41	15:56	16:11	16:26	16:41
MEAN	24.3	29.8	29.5	33.1	30.4	28.8	37.4	25.5	29.2	24.6	26.2	29.5	31.1	28.6	31.1	27.6	34.7
RANGE	9.2	19.1	17.4	12.7	12.6	12.0	10.5	20.0	16.7	16.4	16.4	13.2	16.9	16.2	12.1	19.3	8.1
SUM	121.6	149.1	147.5	165.6	152.1	143.8	187.1	127.6	145.8	123.2	131.1	147.5	155.3	142.9	155.5	138.1	173.4
Cpk	0.188	0.188	0.187	0.188	0.188	0.188	0.190	0.189	0.188	0.186	0.185	0.184	0.184	0.184	0.184	0.183	0.185
Cpm	0.226	0.226	0.225	0.225	0.225	0.226	0.226	0.225	0.225	0.225	0.224	0.224	0.224	0.224	0.224	0.223	0.224
Ppk	0.184	0.183	0.183	0.184	0.184	0.184	0.186	0.184	0.183	0.181	0.180	0.180	0.180	0.179	0.179	0.178	0.180
ALARM	L -	- -	- -	- -	- -	- -	H -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_OUTLINE mode. In the table above, the column outlines in blue and red reflect what is actually displayed in the chart, whereas in the other TableAlarmEmphasisMode examples the outline just shows where the alarm highlighting occurs.

The default mode is ALARM_HIGHLIGHT_NONE mode.

AutoLogAlarmsAsNotes

When an alarm occurs, details of the alarm can be automatically logged as a Notes record. Just set the call the setAutoLogAlarmsAsNotes method.

```
this.setAutoLogAlarmsAsNotes(true);
```

Creating a Batch-Based Variable Control Chart

Both the **SPCTimeVariableContolChart** and **SPCBatchVariableControlChart** derive from the **SPCChartBase** and as a result, the two classes are very similar and share 95% of the same properties. Creating and initializing a batch-based SPC chart is much the same as that of a time-based SPC chart. See the example program BatchVariableControlCharts for a variety of batch SPC charts. Derive your base class from the **SPCBatchVariableControlChart** class.

```
public class BatchXBarRChart extends SPCBatchVariableControlChart {

    GregorianCalendar startTime = new GregorianCalendar();

    //  SPC variable control chart type
        int charttype = SPCControlChartData.MEAN_RANGE_CHART;
    // Number of samples per sub group
        int numsamplespersubgroup = 5;
    // Number of datapoints in the view
        int numdatapointsinview = 17;
    // The time increment between adjacent subgroups
        int timeincrementminutes = 15;


    public BatchXBarRChart()
    {

        // Define and draw chart
        initializeChart();
    }


    public void initializeChart()
    {
        // Initialize the SPCTimeVariableControlChart

        this.initSPCBatchVariableControlChart(charttype,
                numsamplespersubgroup, numdatapointsinview);

.
.
.
        this.rebuildChartUsingCurrentData();
    }
```

Establish the control chart type (Mean Range (X-Bar R), Median Range, X-Bar Sigma (Mean Sigma), Individual Range, EWMA, MA, MAMR, MAMS or CuSum) using the variable control charts **initSPCBatchVariableControlChart** (or **initSPCBatchCusumControlChart** if you are creating a cusum chart) initialization routine. Note that the X-Bar Sigma chart, with a variable subgroup sample size, is initialized using **initSPCBatchVariableControlChart** with a *charttype* value of

MEAN_SIGMA_CHART_VSS. X-Bar Sigma charts with sub groups that use a variable sample size must be updated properly.

SPCBatchVariableControlChart.initSPCBatchVariableControlChart Method

This initialization method initializes the most important values in the creation of a SPC chart.

```
public void initSPCBatchVariableControlChart(
   int charttype,
   int numsamplespersubgroup,
   int numdatapointsinview,
);
```

Parameters

charttype

The SPC chart type parameter. Use one of the SPCControlChartData SPC chart types: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART and MAMS_CHART chart types

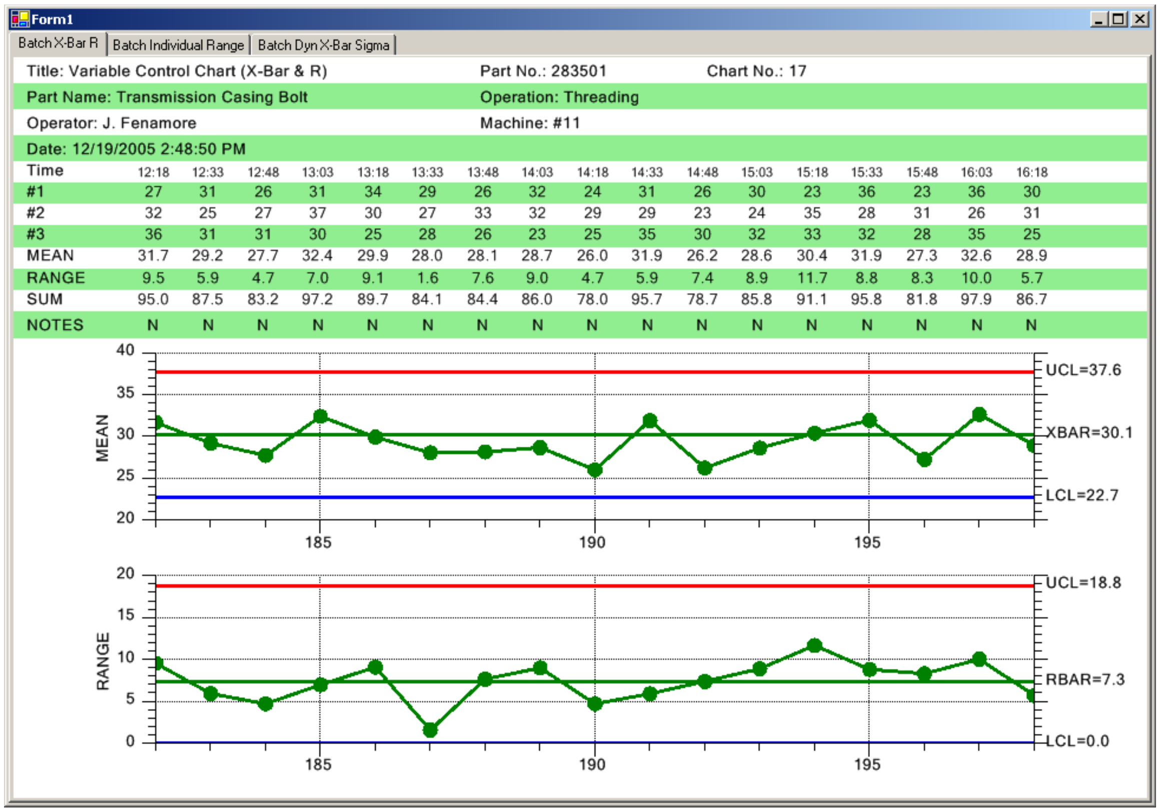
numsamplespersubgroup

Specifies the number of samples that make up a sample subgroup.

numdatapointsinview

Specifies the number of sample subgroups displayed in the graph at one time.

Update the chart data using a **getChartData().addNewSampleRecord** override that has the batch number (*batchCounter* below) as the first parameter. Even though a time stamp value is also used in the **addNewSampleRecord** method, it is not used in the actual graph. Instead, it is used as the time stamp for the batch in the table portion of the chart. The following code is extracted from the BatchVariableControlChart. BatchDynXBarSigmaChart example program**.**



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Part Name: Transmission Casing Bolt Operation: Threading
Operator: J. Fenamore Machine: #11
Date: 12/19/2005 2:48:50 PM

Time	12:18	12:33	12:48	13:03	13:18	13:33	13:48	14:03	14:18	14:33	14:48	15:03	15:18	15:33	15:48	16:03	16:18
#1	27	31	26	31	34	29	26	32	24	31	26	30	23	36	23	36	30
#2	32	25	27	37	30	27	33	32	29	29	23	24	35	28	31	26	31
#3	36	31	31	30	25	28	26	23	25	35	30	32	33	32	28	35	25
MEAN	31.7	29.2	27.7	32.4	29.9	28.0	28.1	28.7	26.0	31.9	26.2	28.6	30.4	31.9	27.3	32.6	28.9
RANGE	9.5	5.9	4.7	7.0	9.1	1.6	7.6	9.0	4.7	5.9	7.4	8.9	11.7	8.8	8.3	10.0	5.7
SUM	95.0	87.5	83.2	97.2	89.7	84.1	84.4	86.0	78.0	95.7	78.7	85.8	91.1	95.8	81.8	97.9	86.7
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

```
private void simulateData()
{
    // Start with 50 sample groups, before start to update dynamically
    for (int i=0; i < 50; i++)
    {
        GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
        // Use the ChartData sample simulator to make an array of sample data
DoubleArray samples = this.getChartData().simulateMeasurementRecord(30, 13);
        // Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord(batchCounter, timestamp, samples);
        // Simulate timeincrementminutes  minute passing
        startTime.add(ChartObj.MINUTE, timeincrementminutes);
        batchCounter++;
    }
}
```

Changing the Batch Control Chart X-Axis Labeling Mode

In revisions prior to 2.0, the x-axis tick marks of a batch control chart could only be labeled with the numeric batch number of the sample subgroup. While batch number labeling is still the default mode, it is now possible to label the sample subgroup tick marks using the time stamp of the sample subgroup, or a user-defined string unique to each sample subgroup.

You may find that labeling every subgroup tick mark with a time stamp, or a user-defined string, causes the axis labels to stagger because there is not enough room to display the tick mark label without overlapping its neighbor. In these cases you may wish to reduce the number of sample subgroups you show on the page using the `numdatapointsinview variable` found in all of the example programs.

```
// Number of datapoints in the view
int numdatapointsinview = 13;
```

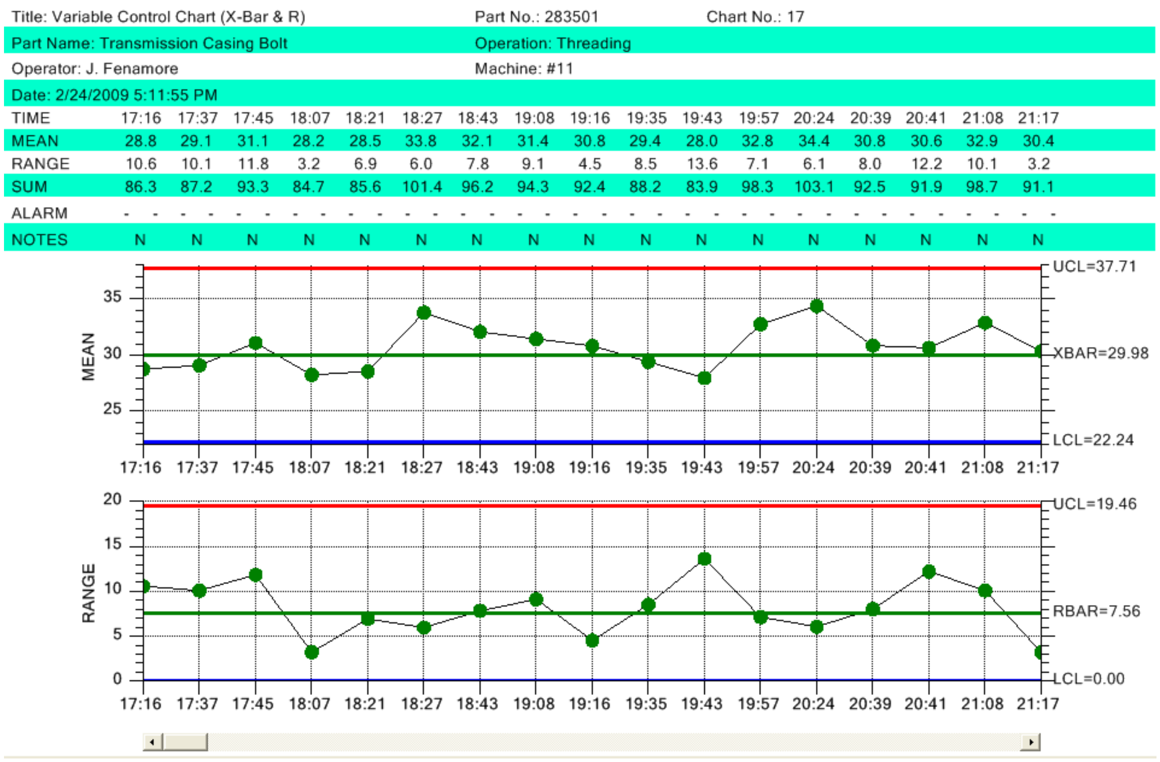
You can rotate the x-axis labels using the charts **setXAxisLabelRotation** method.

```
this.setXAxisLabelRotation( 90);
```

If you rotate the x-axis labels you may need to leave more room between the primary and secondary graphs, and at the bottom, to allow for the increased height of the labels.

```
this.setXAxisLabelRotation(90);

this.setInterGraphMargin(0.1);

this.setGraphBottomPos(0.85);
```

Batch Control Chart X-Axis Time Stamp Labeling



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17
Part Name: Transmission Casing Bolt	Operation: Threading	
Operator: J. Fenamore	Machine: #11	
Date: 2/24/2009 5:11:55 PM		

TIME	17:16	17:37	17:45	18:07	18:21	18:27	18:43	19:08	19:16	19:35	19:43	19:57	20:24	20:39	20:41	21:08	21:17
MEAN	28.8	29.1	31.1	28.2	28.5	33.8	32.1	31.4	30.8	29.4	28.0	32.8	34.4	30.8	30.6	32.9	30.4
RANGE	10.6	10.1	11.8	3.2	6.9	6.0	7.8	9.1	4.5	8.5	13.6	7.1	6.1	8.0	12.2	10.1	3.2
SUM	86.3	87.2	93.3	84.7	85.6	101.4	96.2	94.3	92.4	88.2	83.9	98.3	103.1	92.5	91.9	98.7	91.1
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Batch X-Bar R Chart using time stamp labeling of the x-axis

Set the x-axis labeling mode using the overall charts **setXAxisStringLabelMode** method, setting it `SPCChartObjects.AXIS_LABEL_MODE_TIME`.

```
// enable scroll bar
this.setEnableScrollBar(true);
this.setEnableCategoryValues(false);

// Label the tick mark with time stamp of sample group
this.setXAxisStringLabelMode(SPCChartObjects.AXIS_LABEL_MODE_TIME);
```

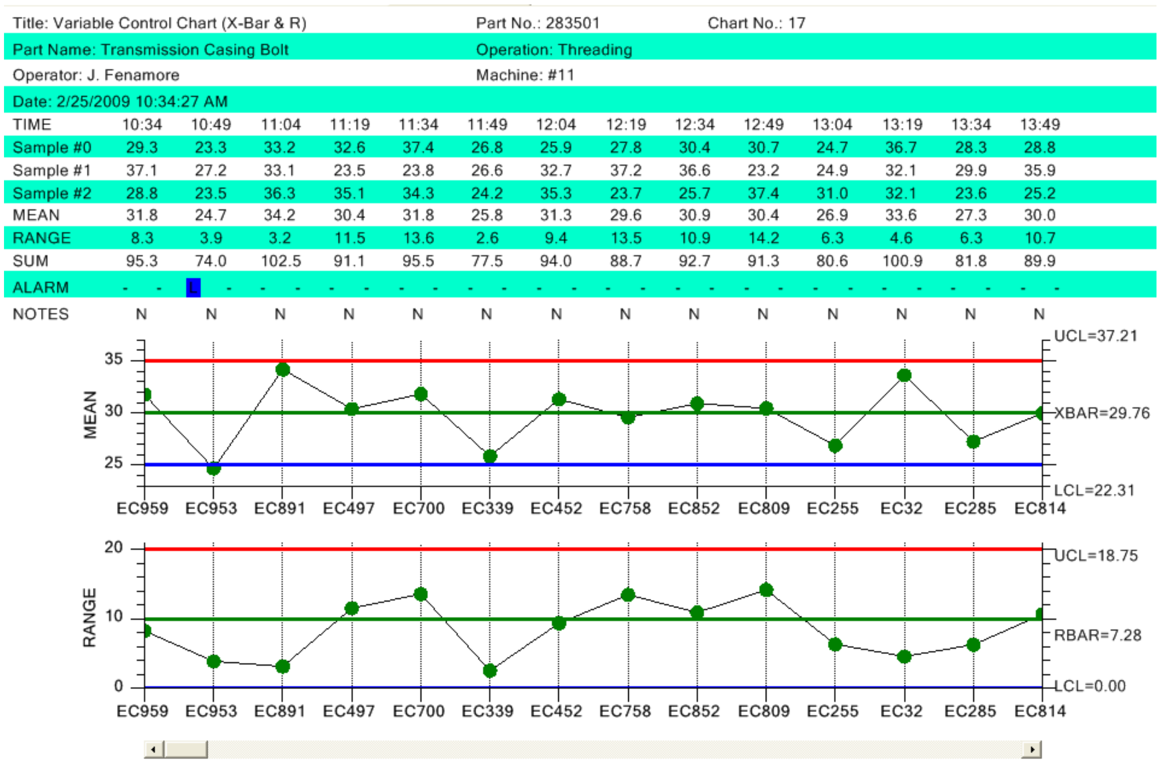
When updating the chart with sample data, use addNewSampleRecord overload which has batch number and a time stamp parameters.

```
this.getChartData().addNewSampleRecord(batchCounter, timestamp, samples);
```

See the example program BatchVariableControlCharts.BatchXBarRChart for a complete example. Reset the axis labeling mode back to batch number labeling by assigning the XAxisStringLabelMode property to SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

```
this.setXAxisStringLabelMode( SPCChartObjects.AXIS_LABEL_MODE_DEFAULT);
```

Batch Control Chart X-Axis User-Defined String Labeling



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading

Operator: J. Fenamore Machine: #11

Date: 2/25/2009 10:34:27 AM

TIME	10:34	10:49	11:04	11:19	11:34	11:49	12:04	12:19	12:34	12:49	13:04	13:19	13:34	13:49
Sample #0	29.3	23.3	33.2	32.6	37.4	26.8	25.9	27.8	30.4	30.7	24.7	36.7	28.3	28.8
Sample #1	37.1	27.2	33.1	23.5	23.8	26.6	32.7	37.2	36.6	23.2	24.9	32.1	29.9	35.9
Sample #2	28.8	23.5	36.3	35.1	34.3	24.2	35.3	23.7	25.7	37.4	31.0	32.1	23.6	25.2
MEAN	31.8	24.7	34.2	30.4	31.8	25.8	31.3	29.6	30.9	30.4	26.9	33.6	27.3	30.0
RANGE	8.3	3.9	3.2	11.5	13.6	2.6	9.4	13.5	10.9	14.2	6.3	4.6	6.3	10.7
SUM	95.3	74.0	102.5	91.1	95.5	77.5	94.0	88.7	92.7	91.3	80.6	100.9	81.8	89.9
ALARM	- -	L -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N

MEAN
35
30
25
UCL=37.21
XBAR=29.76
LCL=22.31
EC959 EC953 EC891 EC497 EC700 EC339 EC452 EC758 EC852 EC809 EC255 EC32 EC285 EC814
RANGE
20
10
0
UCL=18.75
RBAR=7.28
LCL=0.00
EC959 EC953 EC891 EC497 EC700 EC339 EC452 EC758 EC852 EC809 EC255 EC32 EC285 EC814

Batch X-Bar R Chart user-defined string labeling of the x-axis

Set the x-axis labeling mode using the overall charts setXAxisStringLabelMode method, setting it `SPCChartObjects.AXIS_LABEL_MODE_STRING.`

```
// enable scroll bar
this.setEnableScrollBar(true);
this.setEnableCategoryValues(false);


// Label the tick mark with user-defined strings
this.setXAxisStringLabelMode( SPCChartObjects.AXIS_LABEL_MODE_STRING);
```

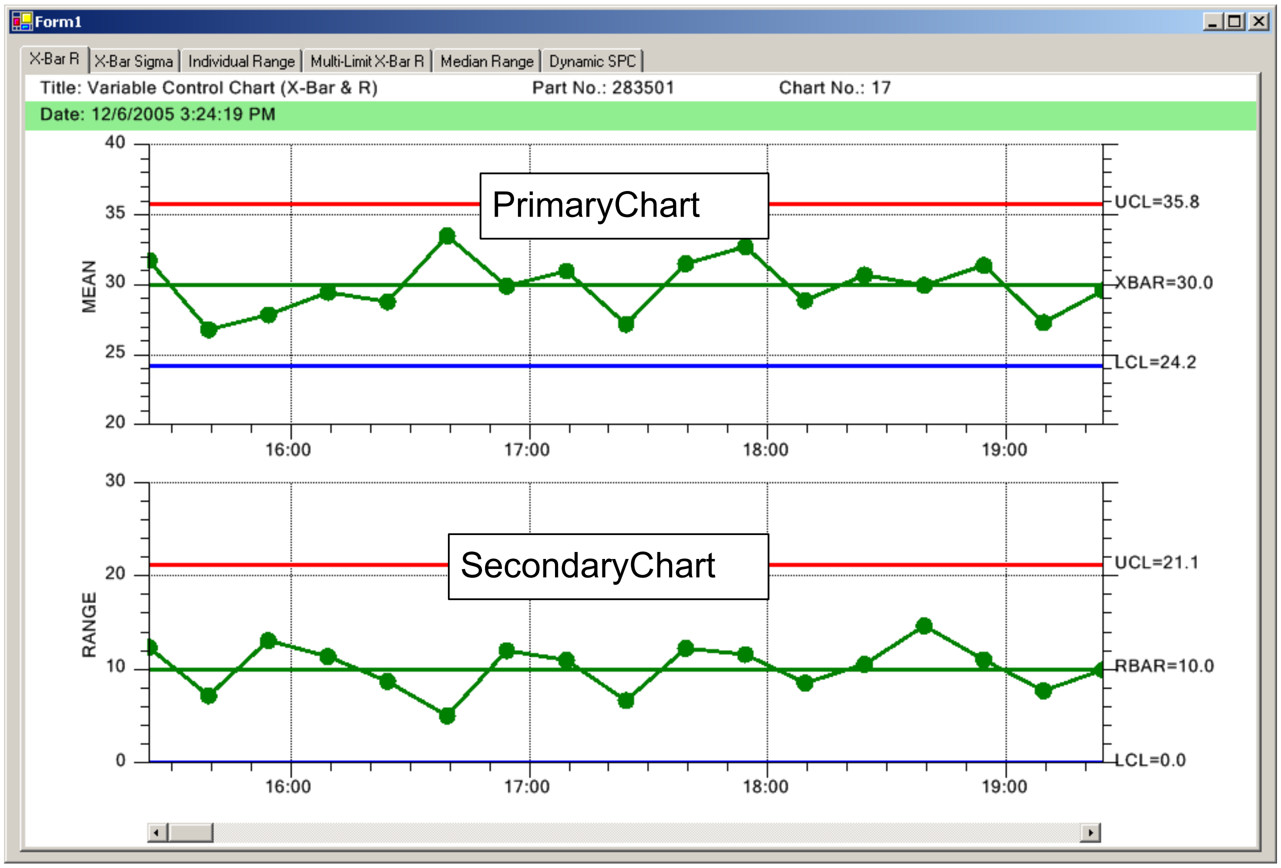
Use the addAxisUserDefinedString method to supply a new string for every new sample subgroup. It must be called every time the addNewSampleRecord method is called, or the user-defined strings will get out of sync with their respective sample subgroup. Reset the axis labeling mode back to batch number labeling by assigning the XAxisStringLabelMode property to SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

```
this.getChartData().addNewSampleRecord(batchCounter, timestamp, samples,
variableControlLimits);


// Make a random string to simulate some sort of batch sample group ID
int randomnum= (int) (1000 * ChartSupport.getRandomDouble());
String batchidstring = "EC" + Integer.toString(randomnum);
this.getChartData().addAxisUserDefinedString(batchidstring);
```

See the example program BatchVariableControlCharts.VariableControlLimits for a complete example.

Changing Default Characteristics of the Chart

All *Variable Control Charts* have two distinct graphs, each with its own set of properties. The top graph is the Primary Chart, and the bottom graph is the Secondary Chart.



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17
Date: 12/6/2005 3:24:19 PM

Logically enough, the properties of the objects that make up each of these graphs are stored in properties named **PrimaryChart** and **SecondaryChart**. Once the graph is initialized (using the **initSPCTimeVariableControlChart**, or **initSPCBatchVariableControlChart** method), you can modify the default characteristics of each graph using these properties.

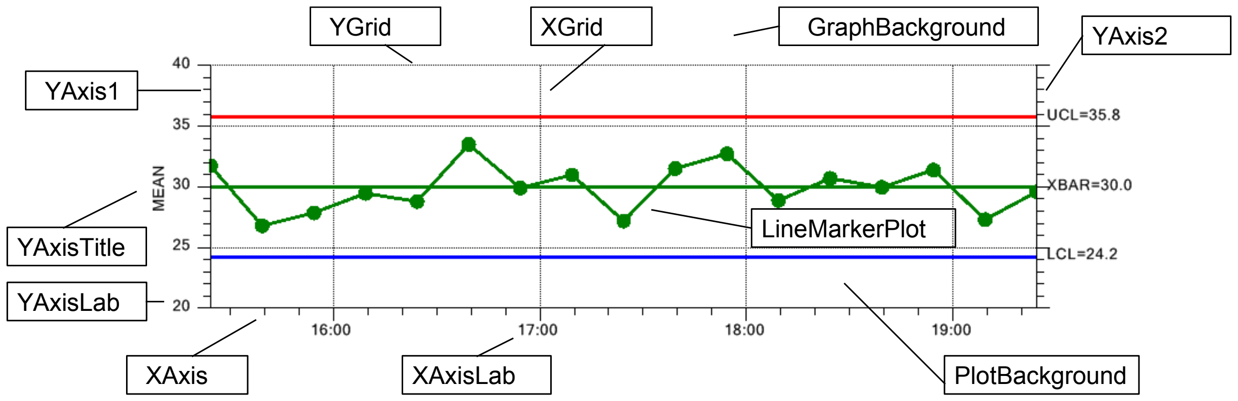
```
SPCChartObjects primaryChart = this.getPrimaryChart();
SPCChartObjects secondaryChart = this.getSecondaryChart();
.
.
.
primaryChart.getXAxis().setLineColor ( ChartColors.BLUE);
primaryChart.getXAxis().setLineWidth ( 1);

secondaryChart.getYAxis1().setLineColor ( ChartColors.GREEN);
secondaryChart.getYAxis2().setLineColor ( ChartColors.RED);
secondaryChart.getYAxis1().setLineWidth ( 1);
secondaryChart.getYAxis2().setLineWidth ( 1);

primaryChart.getProcessVariableData().getLineMarkerPlot().setLineColor (
ChartColors.GREEN);
primaryChart.getProcessVariableData().getLineMarkerPlot().getSymbolAttributes().se
tPrimaryColor ( ChartColors.BLUE);
primaryChart.getProcessVariableData().getLineMarkerPlot().getSymbolAttributes().se
tFillColor ( ChartColors.BEIGE);

primaryChart.getGraphBackground().setFillColor ( ChartColors.LIGHTGRAY);
primaryChart.getPlotBackground().setFillColor ( ChartColors.LIGHTGOLDENRODYELLOW);
secondaryChart.getPlotBackground().setFillColor(ChartColors.LIGHTGOLDENRODYELLOW);
primaryChart.getFrequencyHistogramChart().getPlotBackground().setFillColor (
ChartColors.LIGHTGOLDENRODYELLOW);
secondaryChart.getFrequencyHistogramChart().getPlotBackground().setFillColor (
ChartColors.LIGHTGOLDENRODYELLOW);
```

The **PrimaryChart** and **SecondaryChart** objects are both instances of the **SPCChartObjects** class. The **SPCChartObjects** class contains the objects needed to display a single graph. Below you will find a summary of the class properties.

Class SPCChartObjects – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCChartObjects.html

a link to the JavaDoc documentation for the class **SPCChartObjects**

The main objects of the graph are labeled in the graph below.


YGrid
XGrid
GraphBackground
YAxis2
YAxis1
YAxisTitle
YAxisLab
XAxis
XAxisLab
LineMarkerPlot
PlotBackground
MEAN
40
35
30
25
20
16:00
17:00
18:00
19:00
UCL=35.8
XBAR=30.0
LCL=24.2

7. SPC Attribute Control Charts

SPCTimeAttributeControlChart
SPCBatchAttributeControlChart

Attribute Control Charts are a set of control charts specifically designed for tracking product defects (also called non-conformities). These types of defects are binary in nature (yes/no), where a part has one or more defects, or it doesn't. Examples of defects are paint scratches, discolorations, breaks in the weave of a textile, dents, cuts, etc. Think of the last car that you bought. The defects in each sample group are counted and run through some statistical calculations. Depending on the type of *Attribute Control Chart*, the number of defective parts are tracked (p-chart and np-chart), or alternatively, the number of defects are tracked (u-chart, c-chart). The difference in terminology "number of defective parts" and "number of defects" is highly significant, since a single part not only can have multiple defect categories (scratch, color, dent, etc), it can also have multiple defects per category. A single part may have 0 – N defects. So keeping track of the number of defective parts is statistically different from keeping track of the number of defects. This affects the way the control limits for each chart are calculated.

p-Chart - Also known as the Percent or Fraction Defective Parts Chart
For a sample subgroup, the number of defective parts is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

np-Chart – Also known as the Number Defective Parts Chart
For a sample subgroup, the number of defective parts is measured and plotted as a simple count. Statistically, in order to compare number of defective parts for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

c-Chart - Also known as the Number of Defects or Number of Non-Conformities Chart
For a sample subgroup, the number of times a defect occurs is measured and plotted as a simple count. Statistically, in order to compare number of defects for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

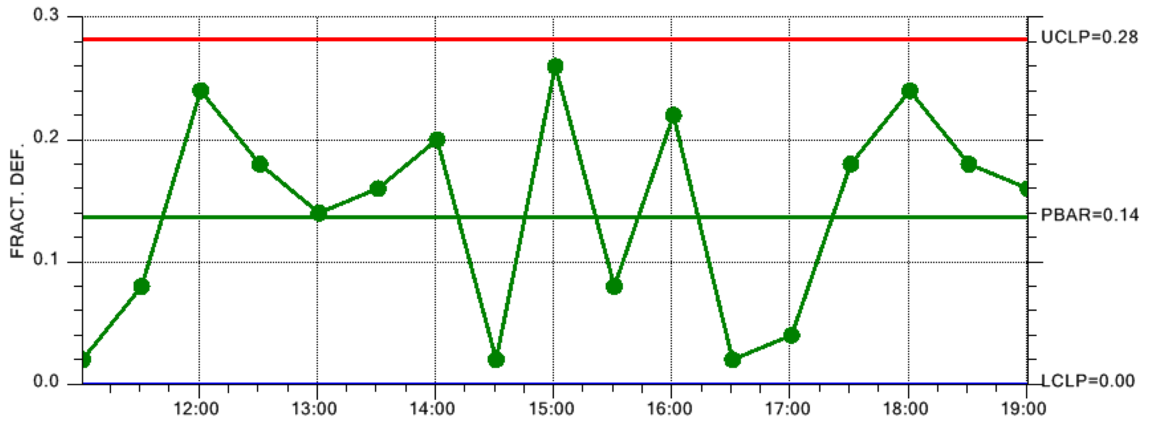
u-Chart – Also known as the Number of Defects per Unit or Number of Non-Conformities per Unit Chart
For a sample subgroup, the number of times a defect occurs is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

DPMO Chart – Also known as the Number of Defects per Million Chart
For a sample subgroup, the number of times a defect occurs is measured and plotted as a value normalized to defects per million. Since the plotted value is normalized to a fixed sample subgroup size, the size of the sample group can vary without rendering the chart useless.

Time-Based and Batch-Based SPC Charts

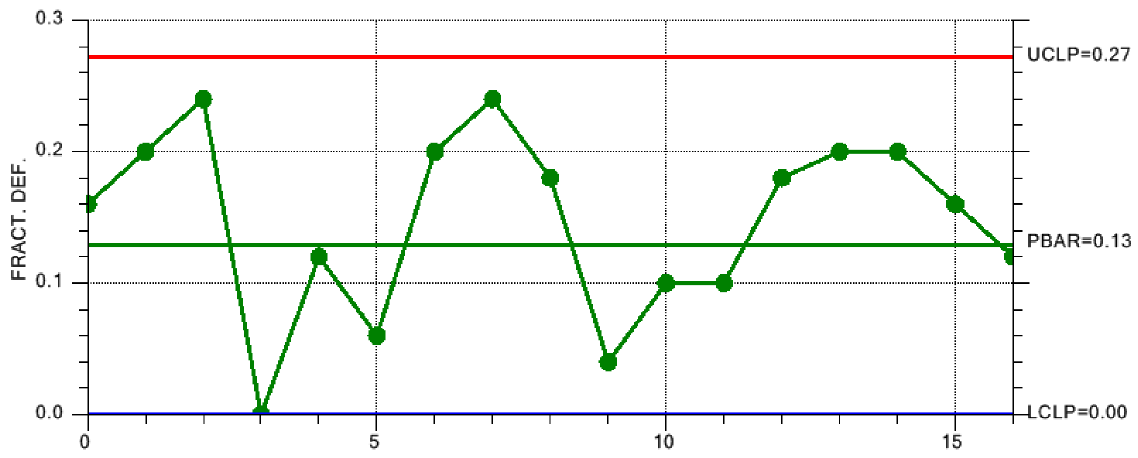
Attribute Control Charts are further categorized as either time- or batch- based. Use time-based SPC charts when data is collected using a subgroup interval corresponding to a specific time interval. Use batch-based SPC charts when the data subgroup interval is a sequential batch number that does not correspond to a uniform time interval. The major difference in these two types of SPC charts is the display of the x-axis. Control charts that sample using a uniform time interval will generally use a time-based x-axis, with time/date axis labels. Control charts that sample based on batches will generally use a numeric-based x-axis, with numeric axis labels.

Time-Based Attribute Control Chart



Note the time-based x-axis.

Batch-Based Attribute Control Chart



Note the numeric based x-axis.

Attribute Control Charts Consist of Only One Graph

Whereas the *Variable Control Charts* contain two different graphs, which we refer to generically as the primary and secondary graphs of the chart, *Attribute Control Charts* only have a single graph, which we refer to generically as the primary graph of the chart.

Creating an Attribute Control Chart

First, select whether you want to use a time-based attribute control chart (use **SPCTimeAttributeControlChart**) or a batch-based attribute control chart (use **SPCBatchAttributeControlChart**). Use that class as the base class for your chart. Since the two classes are very similar and share 95% of all properties in common, only the **SPCTimeAttributeControlChart** is discussed in detail, with the differences between the two classes discussed at the end of the chapter.

```
public class FractionDefectivePartsControlChart extends SPCTimeAttributeControlChart{

        GregorianCalendar startTime = new GregorianCalendar();

        //  SPC attribute control chart type
        int charttype = SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
        // Number of samples per sub group
        int numsamplespersubgroup = 50;
        // Number of defect categories
        int numcategories = 6;
        // Number of datapoints in the view
        int numdatapointsinview = 17;
        // The time increment between adjacent subgroups
        int timeincrementminutes = 30;


        public FractionDefectivePartsControlChart()
        {

                // Define and draw chart
                initializeChart();
        }
        public void initializeChart()
        {
                // Initialize the SPCTimeAttributeControlChart
                this.initSPCTimeAttributeControlChart(charttype, numcategories,
                        numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
                // Change the default horizontal position and width of the chart

.
.
.

                this.rebuildChartUsingCurrentData();


        }
```

SPCTimeAttributeControlChart Members

Class SPCTimeAttributeControlCharts – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCTimeAttributeControlCharts.htm

a link to the JavaDoc documentation for the class **SPCTimeAttributeControlCharts**

The control chart type (p-, np-, c- and u-charts) is established in the attribute control charts **initSPCTimeAttributeControlChart** initialization routine.

SPCTimeAttributeControlChart.initSPCTimeAttributeControlChart Method

This initialization method initializes the most important values in the creation of a SPC chart.

public void initSPCTimeAttributeControlChart(
  int *charttype*,
  int *numcategories*,
  int *numsamplespersubgroup*,
  int *numdatapointsinview*,
  int *timeincrementminutes*
);

Parameters

charttype
Specifies the chart type. Use one of the SPC Attribute Control chart types:
PERCENT_DEFECTIVE_PARTS_CHART,
PERCENT_DEFECTIVE_PARTS_CHART_VSS
FRACTION_DEFECTIVE_PARTS_CHART,
FRACTION_DEFECTIVE_PARTS_CHART_VSS,
NUMBER_DEFECTIVE_PARTS_CHART,
NUMBER_DEFECTS_PERUNIT_CHART,
NUMBER_DEFECTS_PERUNIT_CHART_VSS,

NUMBER_DEFECTS_CHART,
NUMBER_DEFECTS_PER_MILLION_CHART

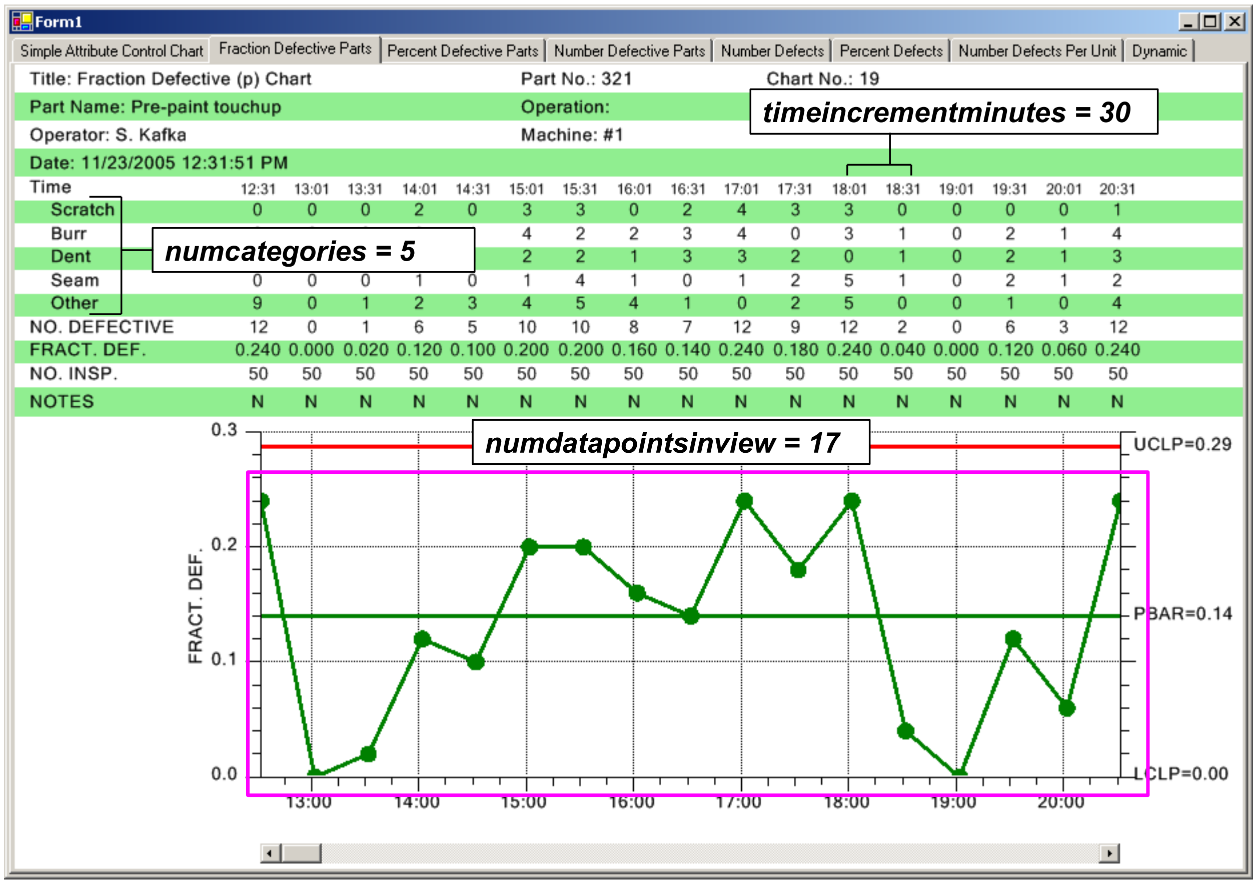
numcategories
In Attribute Control Charts this value represents the number of defect categories used to determine defect counts.

numsamplespersubgroup
In an Attribute Control chart it represents the total sample size per sample subgroup from which the defect data is counted.

numdatapointsinview
Specifies the number of sample subgroups displayed in the graph at one time.

timeincrementminutes
Specifies the normal time increment between adjacent subgroup samples.

The image below further clarifies how these parameters affect the attribute control chart.



Title: Fraction Defective (p) Chart | Part No.: 321 | Chart No.: 19
Part Name: Pre-paint touchup | Operation:
Operator: S. Kafka | Machine: #1
Date: 11/23/2005 12:31:51 PM

Time	12:31	13:01	13:31	14:01	14:31	15:01	15:31	16:01	16:31	17:01	17:31	18:01	18:31	19:01	19:31	20:01	20:31
Scratch	0	0	0	2	0	3	3	0	2	4	3	3	0	0	0	0	1
Burr						4	2	2	3	4	0	3	1	0	2	1	4
Dent						2	2	1	3	3	2	0	1	0	2	1	3
Seam	0	0	0	1	0	1	4	1	0	1	2	5	1	0	2	1	2
Other	9	0	1	2	3	4	5	4	1	0	2	5	0	0	1	0	4
NO. DEFECTIVE	12	0	1	6	5	10	10	8	7	12	9	12	2	0	6	3	12
FRACT. DEF.	0.240	0.000	0.020	0.120	0.100	0.200	0.200	0.160	0.140	0.240	0.180	0.240	0.040	0.000	0.120	0.060	0.240
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Once the init routine is called, the chart can be further customized using properties inherited from **SPCBaseChart**, described below.

Class SPCChartBase - – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCChartBase.html a link to the JavaDoc documentation for the class **SPCChartBase**

Special Note for DPMO Charts

The NUMBER_DEFECTS_PER_MILLION_CHART has an important parameter you may need to set. DPMO charts use an important parameter known is the *defect opportunites per unit*. The default value for the parameter is 1. So if you are using 1 as the the value of *defect opportunites per unit* in your chart, you don't need to do anything. If your value is greater than 1, you need to specify that using code similar to below.

```
this.getChartData().setDefectOpportunitiesPerUnit(5);
```

Adding New Sample Records for Attribute Control Charts.

Attribute Control Chart Cross Reference

p-chart =	FRACTION_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART_VSS, or PERCENT_DEFECTIVE_PARTS_CHART, PERCENT_DEFECTIVE_PARTS_CHART_VSS
np-chart =	NUMBER_DEFECTIVE_PARTS_CHART
c-chart =	NUMBER_DEFECTS_CHART
u-chart =	NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_PERUNIT_CHART_VSS
DPMO =	NUMBER_DEFECTS_PER_MILLION_CHART
p-chart =	FRACTION_DEFECTIVE_PARTS_CHART or PERCENT_DEFECTIVE_PARTS_CHART

Updating p- and np-charts (Fixed Sample Subgroup Size)

p-chart =	FRACTION_DEFECTIVE_PARTS_CHART or PERCENT_DEFECTIVE_PARTS_CHART
np-chart =	NUMBER_DEFECTIVE_PARTS_CHART
DPMO =	NUMBER_DEFECTS_PER_MILLION_CHART

Updating p-, np- and DPMO-charts

In attribute control charts, the meaning of the data in the *samples* array varies, depending on whether the attribute control chart measures the number of defective parts (p-, and np-charts), or the total number of defects (u- and c-charts). The major anomaly is that while the p- and np-charts plot the fraction or number of defective parts, the table portion of the chart can display defect counts for any number of defect categories (i.e. paint scratches, dents, burrs, etc.). It is critical to understand that total number of defects, i.e. the sum of the items in the defect categories for a give sample subgroup, do NOT have to add up to the number of defective parts for the sample subgroup. Every defective part not only can have one or more defects, it can have multiple defects of the same defect category. The total number of defects for a sample subgroup will always be equal to or greater than the number of defective parts. When using p- and np-charts that display defect category counts as part of the table, where N is the *numcategories* parameter in the **InitSPCTimeAttributeControlChart** or **InitSPCBatchAttributeControlChart** initialization call, the first N elements of the *samples* array holds the defect count for each category. The N+1 element of the *samples* array holds the total defective parts count. For example, if you initialized the chart with a *numcategories* parameter to five, signifying that you had five defect categories, you would use a *samples* array sized to six, as in the code below:

```
DoubleArray samples = new DoubleArray(6);
//  GregorianCalendar initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4;    // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5
samples.setElement(5, 4);  // TOTAL number of defective parts in the sample

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord (timestamp, samples);
```

Our example programs obscure this a bit, because we use a special method to simulate defect data for n- and np-charts. The code below is extracted from our TimeAttributeControlCharts.NumberDefectivePartsControlChart example program.

```
DoubleArray samples = this.getChartData().simulateDefectRecord (50 * 0.134,
       SPCControlChartData.NUMBER_DEFECTIVE_PARTS_CHART);
// Add new sample record
this.getChartData().addNewSampleRecord ( timestamp, samples);
```

This particular overload for **getChartData().simulateDefectRecord** knows that since it is a NUMBER_DEFECTIVE_PARTS_CHART chart (np-chart), and since the **ChartData** object was setup with five categories in the **initSPCTimeAttributeControlChart** call, that is should return a **DoubleArray** with (5 + 1 = 6) elements. The first five elements representing simulated defect counts for the five defect categories, and the sixth element the simulated defective parts count. The defect category count data of the *samples* array is only used in the table part of the display; the defect category counts play NO role in the actual SPC chart. The only value plotted in the SPC chart is the last element in the *samples* array, the defective parts count for the sample subgroup.

Updating p-charts (Variable Sample Subgroup Size)

p-chart = FRACTION_DEFECTIVE_PARTS_CHART_VSS
or
PERCENT_DEFECTIVE_PARTS_CHART_VSS

First, you must read the previous section (**Updating p-charts (Fixed Sample Subgroup Size**) and understand it**.** Because in the case of the p-chart variable sample subgroup case, filling out that array is EXACTLY the same as the fixed sample subgroup case. The number of defects in each defect category go into the first N elements (element 0..N-1) of the samples array. The total number of defective parts go into last (element N) of the samples array. Specify the size of the sample subgroup associated with a given update using the **setChartData.SampleSubgroupSize_VSS** method.

```
DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
       SPCControlChartData.NUMBER_DEFECTIVE_PARTS_CHART);

// Randomize the sample subgroup size to some value less than the maximum
// value entered in the call to initSPCTimeAttributeControlChart,
// and set the charts ChartData.SampleSubgroupSize_VSS property with
//  this value immediately prior to the addNewSampleRecord  call.
this.getChartData().setSampleSubgroupSize_VSS(
   numsamplespersubgroup - (int)(25 * ChartSupport.getRandomDouble()));

// Add new sample record
this.getChartData().addNewSampleRecord( timestamp, samples);
```

Updating c- and u-charts (Fixed Sample Subgroup Size)

c-chart = NUMBER_DEFECTS_CHART

u-chart = NUMBER_DEFECTS_PERUNIT_CHART

In c- and u-charts the number of defective parts is of no consequence. The only thing tracked is the number of defects. Therefore, there is no extra array element tacked onto the end of the *samples* array. Each element of the *samples* array represents the total number of defects for a given defect category. If the *numcategories* parameter in the **initSPCTimeAttributeControlChart** or **initSPCBatchAttributeControlChart** is initialized to five, the total number of elements in the *samples* array should be five. For example:

```
DoubleArray samples = new DoubleArray(5);
//  time stamp initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4;   // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord (timestamp, samples);
```

Updating u-charts (Variable Sample Subgroup Size)

u-chart = NUMBER_DEFECTS_PERUNIT_CHART_VSS

First, you must read the previous section (**Updating u-charts Fixed Sample Subgroup Size**) and understand it**.** Because in the case of the u-chart variable sample subgroup case, filling out that array is EXACTLY the same as the fixed sample subgroup case. The number of defects in each defect category go into the first N elements (element 0..N-1) of the samples array. Specify the size of the sample subgroup associated with a given update using the **ChartData.SampleSubgroupSize_VSS** property.

```
DoubleArray samples = new DoubleArray(5);
//  time stamp initialized with current  time by default
GregorianCalendar timestamp = new GregorianCalendar();
```

```
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4;    // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

// Randomize the sample subgroup size to some value less than the maximum
// value entered in the call to initSPCTimeAttributeControlChart,
// and set the charts ChartData.SampleSubgroupSize_VSS property with
//  this value immediately prior to the addNewSampleRecord  call.
this.getChartData().setSampleSubgroupSize_VSS =
    (numsamplespersubgroup - (int)(25 * ChartSupport.getRandomDouble()));

// Add the new sample subgroup to the chart
this.getChartData().addNewSampleRecord(timestamp, samples);
```

While the table portion of the display can display defect data broken down into categories, only the sum of the defects for a given sample subgroup is used in creating the actual SPC chart. Note that the code below, extracted from the TimeAttributeControlCharts.NumberDefectsControlChart example, uses a different **getChartData().simulateDefectRecord** method to simulate the defect data.

```
// Simulate sample record
DoubleArray samples = this.getChartData().simulateDefectRecord (19.85/5);
// Add a sample record
this.getChartData().addNewSampleRecord ( timestamp, samples);
```

Chart Header Information, Measured Data and Calculated Value Table

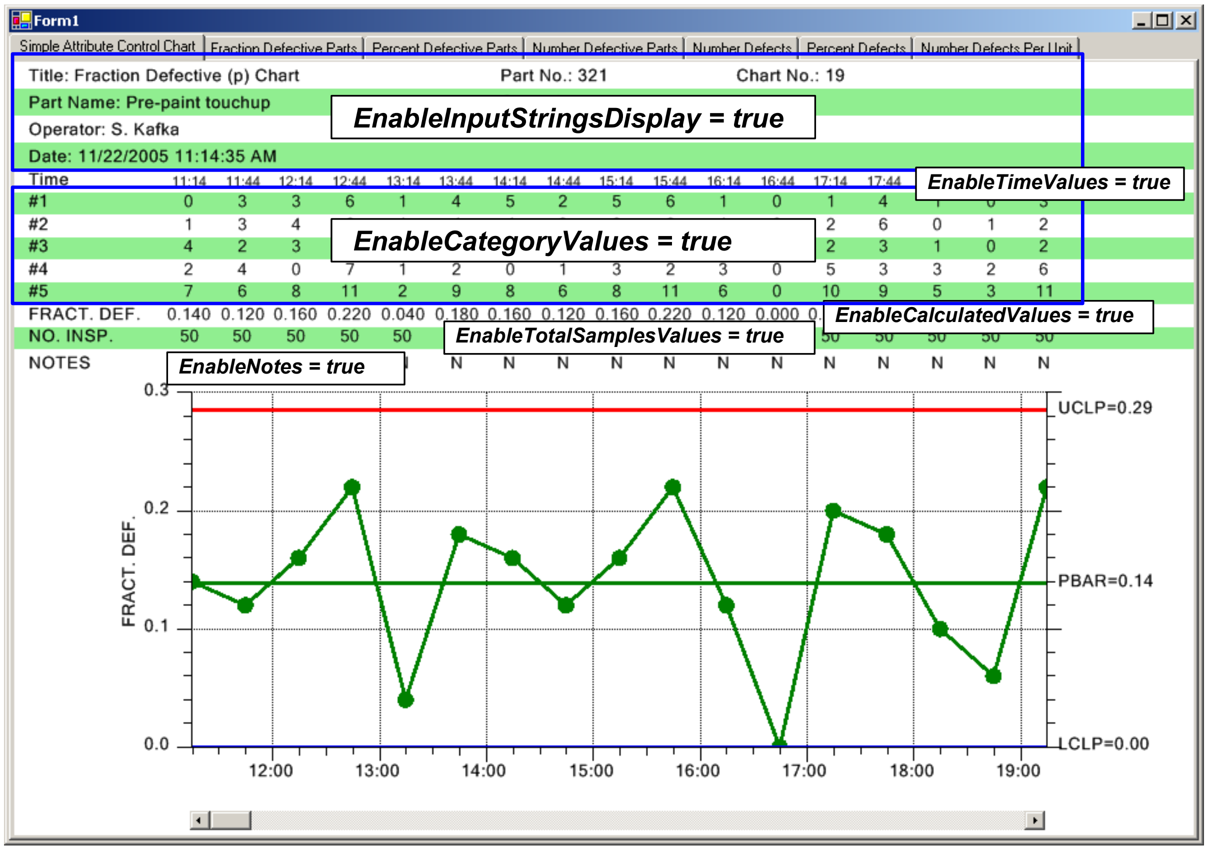
Standard worksheets used to gather and plot SPC data consist of three main parts.

- The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.
- The second part is the measurement data recording and calculation section, organized as a table recording the sample data and calculated values in a neat, readable fashion.
- The third part plots the calculated SPC values as a SPC chart.

The chart includes options that enable the programmer to customize and automatically include header information along with a table of the measurement and calculated data, in the SPC chart.

The following properties enable sections of the chart header and table:

> **setEnableInputStringsDisplay**
> **setEnableCategoryValues**
> **setEnableCalculatedValues**
> **setEnableTotalSamplesValues**
> **setEnableNotes**
> **setEnableTotalSamplesValues**
> **setEnableTimeValues**



Title: Fraction Defective (p) Chart   Part No.: 321   Chart No.: 19
Part Name: Pre-paint touchup
Operator: S. Kafka
Date: 11/22/2005 11:14:35 AM

Time	11:14	11:44	12:14	12:44	13:14	13:44	14:14	14:44	15:14	15:44	16:14	16:44	17:14	17:44	[illegible]	[illegible]	[illegible]
#1	0	3	3	6	1	4	5	2	5	6	1	0	1	4	1	0	3
#2	1	3	4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2	6	0	1	2
#3	4	2	3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2	3	1	0	2
#4	2	4	0	7	1	2	0	1	3	2	3	0	5	3	3	2	6
#5	7	6	8	11	2	9	8	6	8	11	6	0	10	9	5	3	11
FRACT. DEF.	0.140	0.120	0.160	0.220	0.040	0.180	0.160	0.120	0.160	0.220	0.120	0.000	0.	[illegible]	[illegible]	[illegible]	[illegible]
NO. INSP.	50	50	50	50	50	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	50	50	50	50	50
NOTES	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	N	N	N	N	N	N	N	N	N	N	N	N

The example code below is extracted from the TimeAttributeControlCharts.SimpleAttributeControlChart.example program.

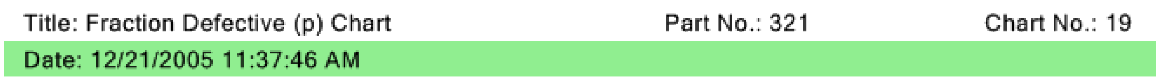
```
public void initializeChart()
{
    // Initialize the SPCTimeAttributeControlChart
```

```
        this.initSPCTimeAttributeControlChart(charttype, numcategories,
            numsamplespersubgroup, numdatapointsinview, timeincrementminutes);

        // Change the default horizontal position and width of the chart

        // Set the strings used in the header section of the table
        SPCControlChartData chartdata = this.getChartData();

        chartdata.setTitle("Fraction Defective (p) Chart");
        chartdata.setPartNumber ( "321");
        chartdata.setChartNumber("19");
        chartdata.setPartName( "Pre-paint touchup");
        chartdata.setTheOperator("B. Cornwall");
        chartdata.setPartName( "Left Front Fender");
        chartdata.setOperation ( "Painting");
        chartdata.setSpecificationLimits("");
        chartdata.setMachine("#11");
        chartdata.setGage("");
        chartdata.setUnitOfMeasure ( "");
        chartdata.setZeroEquals("");
        chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));

        this.setHeaderStringsLevel ( SPCControlChartData.HEADER_STRINGS_LEVEL3);

        // Display the Sampled value rows of the table
        this.setEnableInputStringsDisplay( true);
        // Display the Sampled value rows of the table
        this.setEnableCategoryValues( true);
        // Display the Calculated value rows of the table
        this.setEnableCalculatedValues( true);
        // Display the total samples per subgroup value row
        this.setEnableTotalSamplesValues( true);
        // Display the Notes row of the table
        this.setEnableNotes( true);
        // Display the time stamp row of the table
        this.setEnableTimeValues ( true);
.
.
.
        this.rebuildChartUsingCurrentData();


    }
```

The input header strings display has four sub-levels that display increasing levels of information. The input header strings display level is set using the charts HeaderStringsLevel property. Strings that can be displayed are: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString. The four levels and the information displayed is listed below:

HEADER_STRINGS_LEVEL0	Display no header information
HEADER_STRINGS_LEVEL1	Display minimal header information: Title, PartNumber, ChartNumber, DateString
HEADER_STRINGS_LEVEL2	Display most header strings: Title, PartNumber, ChartNumber, PartName, Operation, Operator, Machine, DateString
HEADER_STRINGS_LEVEL3	Display all header strings: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString

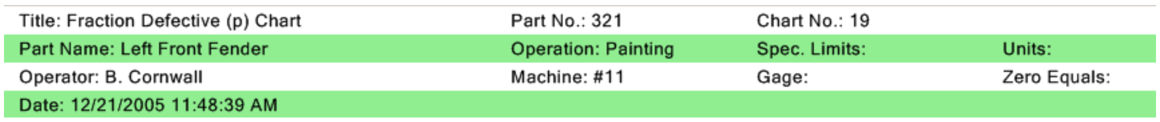
The example program TimeAttributeControlCharts.SimpleAttributeControlChart demonstrates the use of the **HeaderStringsLevel** property. The example below displays a minimum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL1).



Title: Fraction Defective (p) Chart	Part No.: 321	Chart No.: 19
Date: 12/21/2005 11:37:46 AM		

```
// Set the strings used in the header section of the table
SPCControlChartData chartdata = this.getChartData();

chartdata.setTitle("Fraction Defective (p) Chart");
chartdata.setPartNumber ( "321");
chartdata.setChartNumber("19");
chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));
this.setHeaderStringsLevel( SPCControlChartData.HEADER_STRINGS_LEVEL1);
```

The example below displays a maximum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL3).



Title: Fraction Defective (p) Chart	Part No.: 321	Chart No.: 19	
Part Name: Left Front Fender	Operation: Painting	Spec. Limits:	Units:
Operator: B. Cornwall	Machine: #11	Gage:	Zero Equals:
Date: 12/21/2005 11:48:39 AM			

```
// Set the strings used in the header section of the table
SPCControlChartData chartdata = this.getChartData();

chartdata.setTitle("Fraction Defective (p) Chart");
chartdata.setPartNumber ( "321");
chartdata.setChartNumber("19");
chartdata.setPartName( "Pre-paint touchup");
chartdata.setTheOperator("B. Cornwall");
chartdata.setPartName( "Left Front Fender");
chartdata.setOperation ( "Painting");
chartdata.setSpecificationLimits("");
chartdata.setMachine("#11");
chartdata.setGage("");
chartdata.setUnitOfMeasure ( "");
chartdata.setZeroEquals("");
chartdata.setDateString( ChartCalendar.toString(new GregorianCalendar()));

this.setHeaderStringsLevel ( SPCControlChartData.HEADER_STRINGS_LEVEL3);
```

The identifying string displayed in front of the input header string can be any string that you want, including non-English language string. For example, if you want the input header string for the Title to represent a project name:

Project Name: Project XKYZ for PerQuet

Set the properties:

```
this.getChartData().setTitle("Project XKYZ for PerQuet");
this.getChartData().setTitleHeader("Project Name:");
```

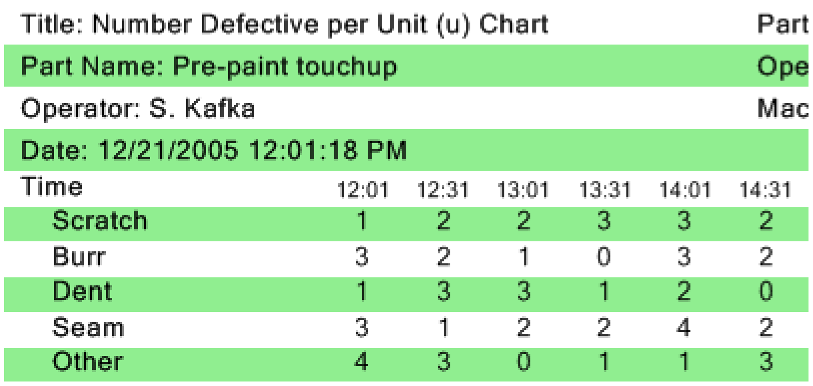
Change other headers using the ChartData properties listed below.

- TitleHeader
- PartNumberHeader
- ChartNumberHeader
- PartNameHeader
- OperationHeader
- OperatorHeader
- MachineHeader
- DateHeader
- SpecificationLimitsHeader
- GageHeader

⑤ UnitOfMeasureHeader
⑤ ZeroEqualsHeader
⑤ NotesHeader

Even though the input header string properties have names like Title, PartNumber, ChartNumber, etc., those names are arbitrary. They are really just placeholders for the strings that are placed at the respective position in the table. You can display any combination of strings that you want, rather than the ones we have selected by default, based on commonly used standardized SPC Control Charts.

Depending on the control chart type, you may want to customize the category header strings. In most of our examples, we use the category header strings: Scratch, Burr, Dent, Seam, and Other, to represent common defect categories. You can change these strings to anything that you want using the **getChartData().setSampleRowHeaderString** method. See the example program TimeAttributeControlCharts.NumberDefectsControlChart.



Title: Number Defective per Unit (u) Chart — Part
Part Name: Pre-paint touchup — Ope
Operator: S. Kafka — Mac
Date: 12/21/2005 12:01:18 PM

Time	12:01	12:31	13:01	13:31	14:01	14:31
Scratch	1	2	2	3	3	2
Burr	3	2	1	0	3	2
Dent	1	3	3	1	2	0
Seam	3	1	2	2	4	2
Other	4	3	0	1	1	3

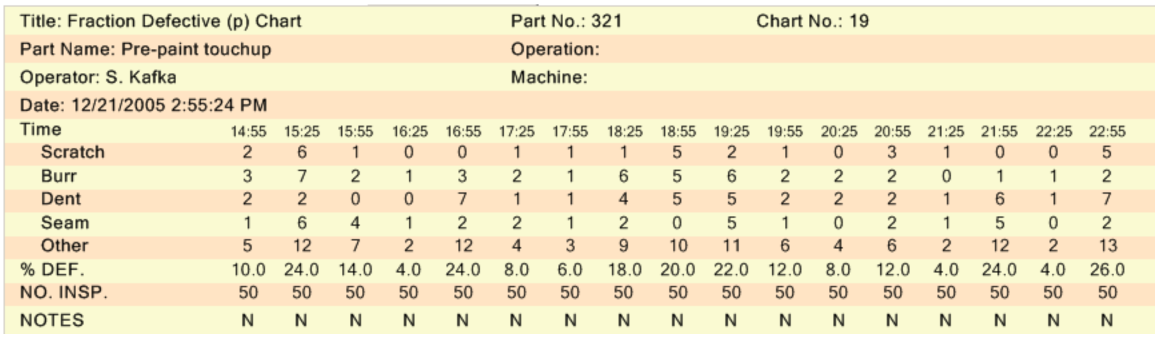
```
// Set the table row headers strings for defect categories
SPCControlChartData chartdata = this.getChartData();

chartdata.setSampleRowHeaderString(0, "    Scratch");
chartdata.setSampleRowHeaderString(1, "    Burr");
chartdata.setSampleRowHeaderString(2, "    Dent");
chartdata.setSampleRowHeaderString(3, "    Seam");
chartdata.setSampleRowHeaderString(4, "    Other");
```

The **ChartTable** property of the chart has properties that further customize the chart. The default table background uses the accounting style green-bar striped background. You can change this using the **ChartTable.TableBackgroundMode** property**.** Set the value to one of the TableBackgroundMode constants:

TABLE_NO_COLOR_BACKGROUND	Constant specifies that the table does not use a background color.

TABLE_SINGLE_COLOR_BACKGROUND	Constant specifies that the table uses a single color for the background (backgroundColor1)
TABLE_STRIPED_COLOR_BACKGROUND	Constant specifies that the table uses horizontal stripes of color for the background (backgroundColor1 and backgroundColor2)

Extracted from the TimeAttributeControlCharts.PercentDefectivePartsControlChart example program



Title: Fraction Defective (p) Chart　Part No.: 321　Chart No.: 19

Part Name: Pre-paint touchup　Operation:

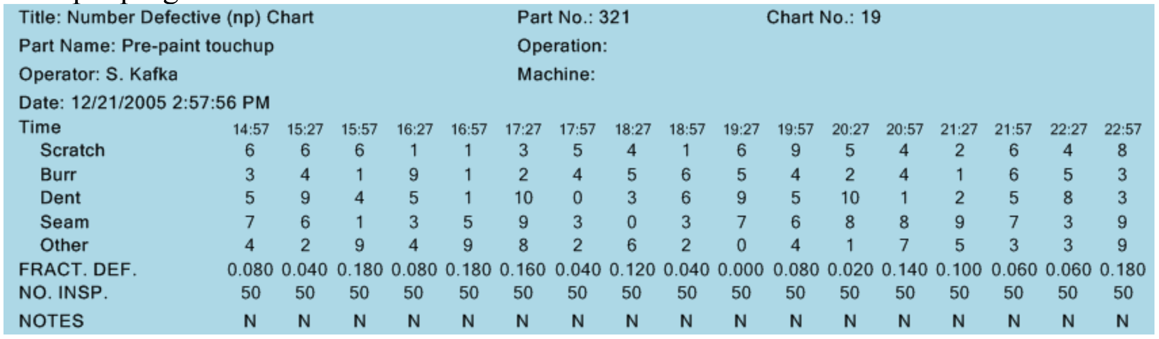
Operator: S. Kafka　Machine:

Date: 12/21/2005 2:55:24 PM

Time	14:55	15:25	15:55	16:25	16:55	17:25	17:55	18:25	18:55	19:25	19:55	20:25	20:55	21:25	21:55	22:25	22:55
Scratch	2	6	1	0	0	1	1	1	5	2	1	0	3	1	0	0	5
Burr	3	7	2	1	3	2	1	6	5	6	2	2	2	0	1	1	2
Dent	2	2	0	0	7	1	1	4	5	5	2	2	2	1	6	1	7
Seam	1	6	4	1	2	2	1	2	0	5	1	0	2	1	5	0	2
Other	5	12	7	2	12	4	3	9	10	11	6	4	6	2	12	2	13
% DEF.	10.0	24.0	14.0	4.0	24.0	8.0	6.0	18.0	20.0	22.0	12.0	8.0	12.0	4.0	24.0	4.0	26.0
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

```
SPCGeneralizedTableDisplay chartTable = this.getChartTable();


chartTable.setTableBackgroundMode(
    SPCGeneralizedTableDisplay.TABLE_STRIPED_COLOR_BACKGROUND);
chartTable.setBackgroundColor1( ChartColors.BISQUE);
chartTable.setBackgroundColor2( ChartColors.LIGHTGOLDENRODYELLOW);
```

Extracted from the TimeAttributeControlCharts.NumberDefectivePartsControlChart example program



Title: Number Defective (np) Chart　Part No.: 321　Chart No.: 19

Part Name: Pre-paint touchup　Operation:

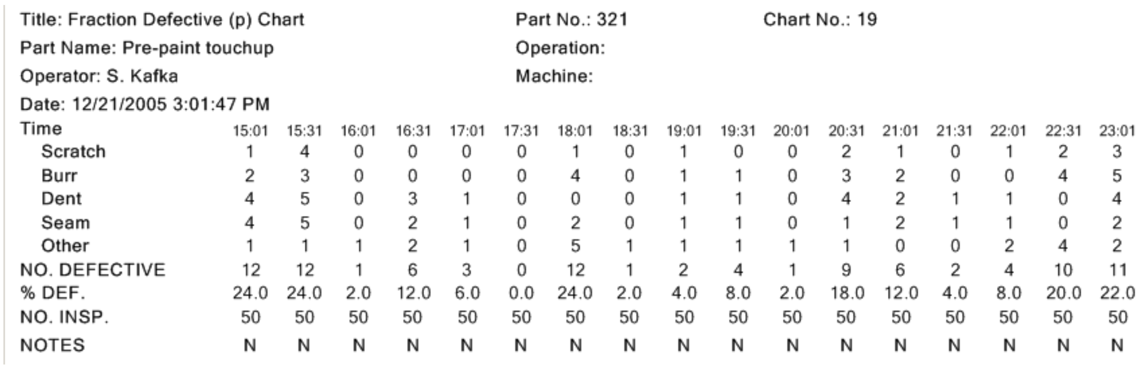
Operator: S. Kafka　Machine:

Date: 12/21/2005 2:57:56 PM

Time	14:57	15:27	15:57	16:27	16:57	17:27	17:57	18:27	18:57	19:27	19:57	20:27	20:57	21:27	21:57	22:27	22:57
Scratch	6	6	6	1	1	3	5	4	1	6	9	5	4	2	6	4	8
Burr	3	4	1	9	1	2	4	5	6	5	4	2	4	1	6	5	3
Dent	5	9	4	5	1	10	0	3	6	9	5	10	1	2	5	8	3
Seam	7	6	1	3	5	9	3	0	3	7	6	8	8	9	7	3	9
Other	4	2	9	4	9	8	2	6	2	0	4	1	7	5	3	3	9
FRACT. DEF.	0.080	0.040	0.180	0.080	0.180	0.160	0.040	0.120	0.040	0.000	0.080	0.020	0.140	0.100	0.060	0.060	0.180
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

```
this.getChartTable().setTableBackgroundMode(
    SPCGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND);
this.getChartTable().setBackgroundColor1(ChartColors.LIGHTBLUE);
```

Extracted from the TimeAttributeControlCharts.NumberDefectivePartsControlChart example program



Title: Fraction Defective (p) Chart　　Part No.: 321　　Chart No.: 19
Part Name: Pre-paint touchup　　Operation:
Operator: S. Kafka　　Machine:
Date: 12/21/2005 3:01:47 PM

Time	15:01	15:31	16:01	16:31	17:01	17:31	18:01	18:31	19:01	19:31	20:01	20:31	21:01	21:31	22:01	22:31	23:01
Scratch	1	4	0	0	0	0	1	0	1	0	0	2	1	0	1	2	3
Burr	2	3	0	0	0	0	4	0	1	1	0	3	2	0	0	4	5
Dent	4	5	0	3	1	0	0	0	1	1	0	4	2	1	1	0	4
Seam	4	5	0	2	1	0	2	0	1	1	0	1	2	1	1	0	2
Other	1	1	1	2	1	0	5	1	1	1	1	1	0	0	2	4	2
NO. DEFECTIVE	12	12	1	6	3	0	12	1	2	4	1	9	6	2	4	10	11
% DEF.	24.0	24.0	2.0	12.0	6.0	0.0	24.0	2.0	4.0	8.0	2.0	18.0	12.0	4.0	8.0	20.0	22.0
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

```
this.getChartTable().setTableBackgroundMode(
        SPCGeneralizedTableDisplay.TABLE_NO_COLOR_BACKGROUND);
```

Table and Chart Fonts

There are a large number of fonts that you have control over, both the fonts in the table and the fonts in the chart. The programmer can select a default font (as in the case of non-US character set), or select individual fonts for different elements of the table and charts.

Table Fonts

The table fonts are accessed through the charts **ChartTable** property. Below is a list of accessible table fonts:

TimeLabelFont	The font used in the display of time values in the table.
SampleLabelFont	The font used in the display of sample numeric values in the table.
CalculatedLabelFont	The font used in the display of calculated values in the table.
StringLabelFont	The font used in the display of header string values in the table.
NotesLabelFont	The font used in the display of notes values in the table.

Extracted from the example BatchAttributeControlCharts.PercentDefectivePartsControlChart

```
this.getChartTable().setSampleLabelFont (new Font("Serif", Font.PLAIN, 12));
```

The **ChartTable** class has a static property, **SPCGeneralizedTableDisplay.DefaultTableFont**, that sets the default Font. Use this if you want to establish a default font for all of the text in a table. This static property must be set BEFORE the charts **init** routine.

Extracted from the example
BatchAttributeControlCharts.PercentDefectivePartsControlChart

```
SPCGeneralizedTableDisplay.setDefaultTableFont(
       new Font("Serif",  Font.PLAIN, 12));
// Initialize the SPCBatchVariableControlChart
this.initSPCBatchAttributeControlChart(charttype, numcategories,
               numsamplespersubgroup, numdatapointsinview);
.
.
.
```

Chart Fonts
There are default chart fonts that are static objects in the **SPCChartObjects** class. They establish the default fonts for related chart objects and if you change them they need to be set before the first charts Init.. call. Since these properties are static, any changes to them will apply to the program as a whole, not just the immediate class.

AxisLabelFont	The font used to label the x- and y- axes.
AxisTitleFont	The font used for the axes titles.
HeaderFont	The font used for the chart title.
SubheadFont	The font used for the chart subhead.
ToolTipFont	The tool tip font.
AnnotationFont	The annotation font.
ControlLimitLabelFont	The font used to label the control limits

Extracted from the example TimeAttributeControlCharts.PercentDefectiveChart

```
SPCChartObjects.setAxisTitleFont( new Font("Serif", Font.PLAIN, 12));
SPCChartObjects.setControlLimitLabelFont( new Font("Serif", Font.PLAIN, 12));


this.initSPCTimeAttributeControlChart(charttype, numcategories,
               numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
```

The chart class static property, **DefaultTableFont**, sets the default Font string. Since the chart fonts all default to different sizes, the default font is defined using a string specifying the name of the font. This static property must be set BEFORE the charts **init** routine.

Extracted from the example Extracted from the example
TimeAttributeControlCharts.PercentDefectiveChart

```
SPCTimeVariableControlChart.setDefaultChartFontString( "Times");
this.initSPCTimeAttributeControlChart(charttype, numcategories,
               numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
.
.
.


.
.
```

These static properties establish the default fonts for a group of objects as a whole. For example, all charts will have the same x- and y-axis label fonts. You can still change the individual fonts for an individual object in a specific chart. For example, if in the Primary Chart you want the x-axis label font to be size 10, and the y-axis label font to be size 14, you can set them individually after the charts Init.. method has been called.

```
this.initSPCTimeAttributeControlChart(charttype, numcategories,
               numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
.
.
this.getPrimaryChart().getXAxisLab().setTextFont(new Font("Serif", Font.PLAIN,
12));
this.getPrimaryChart().getYAxisLab().setTextFont new Font("Serif", Font.PLAIN,
12));
```

Table and Chart Templates

All of the strings displayed in the table and charts use a template unique to the string type. Numeric strings use a **NumericLabel** template, time/date strings use a time **TimeLabel** template, and so on. These templates permit the programmer to customize the display of the strings. The various templates are listed below:

SPCChartObjects (Accessed in the charts PrimaryChart and SecondaryChart properties)

Propertry	Type	Description
XValueTemplate	NumericLabel	The x-value template for the data tooltip.
YValueTemplate	NumericLabel	The y-value template for the data tooltip.
XTimeValueTemplate	TimeLabel	x-value template for the data tooltip.
TextTemplate	ChartText	The text template for the data tooltip.

SPCGeneralizedTableDisplay (Accessed in the charts ChartTable property)

Propertry	**Type**	**Description**
TimeItemTemplate	TimeLabel	The TimeLabel object used as a template for displaying time values in the table.
SampleItemTemplate	NumericLabel	The NumericLabel object used as a template for displaying the sample values in the table.
CalculatedItemTemplate	NumericLabel	The NumericLabel object used as a template for displaying calculated values in the table.
StringItemTemplate	StringLabel	The StringLabel object used as a template for displaying string values in the table.
NotesItemTemplate	NotesLabel	The NotesLabel object used as a template for displaying string values in the table.

The most common use for these templates is to set the color attributes of a class of objects, or the decimal precision of a numeric string.

```
this.getChartTable().getSampleItemTemplate().setLineColor(ChartColors.RED);
```

Chart Position

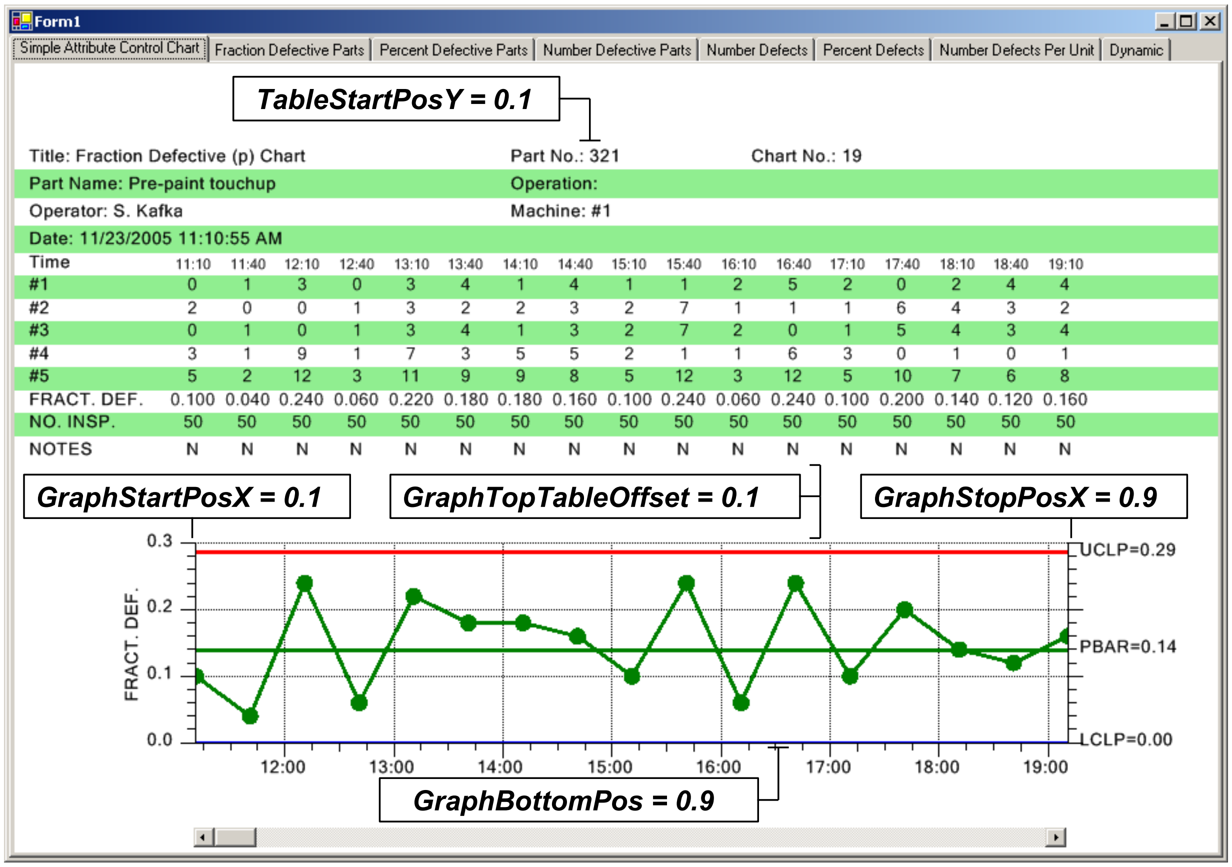
If the SPC chart does not include frequency histograms on the left (they take up about 20% of the available chart width), you may want to adjust the left and right edges of the chart using the **GraphStartPosX** and **GraphStopPlotX** properties to allow for more room in the display of the data. This also affects the table layout, because the table columns must line up with the chart data points.

```
this.setGraphStartPosX( 0.1); // start here
this.setGraphStopPosX( 0.875);  // end here
```

There is not much flexibility positioning the top and bottom of the chart. Depending on the table items enabled, the table starts at the position defined by the **TableStartPosY** property, and continues until all of the table items are displayed. It then offsets the top of the primary chart with respect to the bottom of the table by the value of the property **GraphTopTableOffset**. The value of the property **GraphBottomPos** defines the bottom of the graph. The default values for these properties are:

```
this.setTableStartPosY( 0.00);
this.setGraphTopTableOffset( 0.02);
this.setGraphBottomPos( 0.925);
```

The picture below uses different values for these properties in order to emphasize the affect that these properties have on the resulting chart.



Title: Fraction Defective (p) Chart   Part No.: 321   Chart No.: 19

Part Name: Pre-paint touchup   Operation:

Operator: S. Kafka   Machine: #1

Date: 11/23/2005 11:10:55 AM

Time	11:10	11:40	12:10	12:40	13:10	13:40	14:10	14:40	15:10	15:40	16:10	16:40	17:10	17:40	18:10	18:40	19:10
#1	0	1	3	0	3	4	1	4	1	1	2	5	2	0	2	4	4
#2	2	0	0	1	3	2	2	3	2	7	1	1	1	6	4	3	2
#3	0	1	0	1	3	4	1	3	2	7	2	0	1	5	4	3	4
#4	3	1	9	1	7	3	5	5	2	1	1	6	3	0	1	0	1
#5	5	2	12	3	11	9	9	8	5	12	3	12	5	10	7	6	8
FRACT. DEF.	0.100	0.040	0.240	0.060	0.220	0.180	0.180	0.160	0.100	0.240	0.060	0.240	0.100	0.200	0.140	0.120	0.160
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

SPC Control Limits

There are two methods you can use to set the SPC control limit for a chart. The first method explicitly sets the limits to values that you calculate on your own, because of some analysis that a quality engineer does on previously collected data. The second method auto-calculates the limits using the algorithms supplied in this software.

The quick way to set the limit values and limit strings is to use the charts **getChartData().setControlLimitValues** and **getChartData().setControlLimitStrings** methods. This method only works for the default +-3-sigma level control limits, and not any others you may have added using the charts **addAdditionalControlLimit** method discussed in the *Multiple Control Limits* section. The data values in the *controllimitvalues* and *controllimitstrings* arrays used to pass the control limit information must be sorted in the following order:

[SPC_PRIMARY_CONTROL_TARGET,
SPC_PRIMARY_LOWER_CONTROL_LIMIT,
SPC_PRIMARY_UPPER_CONTROL_LIMIT]

```
double [] controllimitvalues = {0.13, 0.0, 0.25};
this.getChartData().setControlLimitValues (controllimitvalues);
```

```
String [] controllimitstrings = {"PBAR","LCL", "UCL”};
this.getChartData().setControlLimitStrings(controllimitstrings);
```

You can also set the control limit values and control limit text one value at a time using the **getChartData().setControlLimitValue** and **getChartData().setControlLimitString** methods.

A more complicated way to set the control limits explicitly is to first grab a reference to the **SPCControlLimitRecord** for a given control limit, and then change the value of that control limit, and the control limit text, if desired. The example below sets the control limit values and text for the three control limits (target value, upper control limit, and lower control limit) of the primary chart, and the three control limit values for the secondary chart.

```
// Set control limits for primary chart

//target control limit primary chart
SPCControlLimitRecord primarytarget =
    getChartData().gtControlLimitRecord(SPCControlChartData.SPC_PRIMARY_CONTROL_TARGET);
primarytarget.setControlLimitValue(0.13);
primarytarget.setControlLimitText("PBAR");

//lower control limit primary chart
SPCControlLimitRecord primarylowercontrollimit =
getChartData().getControlLimitRecord(SPCControlChartData.SPC_PRIMARY_LOWER_CONTROL_LIMIT);
primarylowercontrollimit.setControlLimitValue( 0.0);
primarylowercontrollimit.setControlLimitText( "LCL");

//upper control limit primary chart
SPCControlLimitRecord primaryuppercontrollimit =
getChartData().getControlLimitRecord(SPCControlChartData.SPC_PRIMARY_UPPER_CONTROL_LIMIT);
primaryuppercontrollimit.setControlLimitValue( 0.25);
primaryuppercontrollimit.setControlLimitText("UCL");
```

The second way to set the control limits is to call the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

```
// Must have data loaded before any of the Auto.. methods are called
simulateData();
```

```
// Calculate the SPC control limits for both graphs of the current SPC

this.autoCalculateControlLimits ();
```

You can add data to the **ChartData** object, auto-calculate the control limits to establish the SPC control limits, and then continue to add new data values. Alternatively, you can set the SPC control limits explicitly as the result of previous runs, using the previously described **getChartData().setControlLimitValues** method, and add new sampled data values to the **ChartData** object, and after a certain number of updates call the **autoCalculateControlLimits** method to establish new control limits.

```
updateCount++;
this.getChartData().addNewSampleRecord (timestamp, samples);
if (updateCount > 50) // After 50 sample groups and calculate limits on the fly
{
// Calculate the SPC control limits for the X-Bar part of the current SPC chart
    this.autoCalculateControlLimits ();
    // Scale the y-axis of the X-Bar chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
}
```

Need to exclude records from the control limit calculation? Call the **getChartData().excludeRecordFromControlLimitCalculations** method, passing in true to exclude the record.

```
for (int i=0; i < 10; i++)
      this.getChartData().ExcludeRecordFromControlLimitCalculations(i,true);
```

Formulas Used in Calculating Control Limits for Attribute Control Charts

The SPC control limit formulas used in the software derive from the following source:

Fraction Defective Parts, Number Defective Parts, Number Defects, Number Defects Per Unit - “Introduction to Statistical Quality Control” by Douglas C. Montgomery, John Wiley and Sons, Inc. 2001.

Percent Defective Parts - “SPC Simplified – Practical Steps to Quality” by Robert T. Amsden, Productivity Inc., 1998.

SPC Control Chart Nomenclature

UCL = Upper Control Limit

LCL = Lower Control Limit

Center line = The target value for the process

p = estimate (or average) of the fraction defective (or non-conforming) parts

P = estimate (or average) of the percent defective (or non-conforming) parts

c = estimate (or average) of the number of defects (or nonconformities)

u = estimate (or average) of the number of defects (or nonconformities) per unit

n = number of samples per subgroup

dopu = defect opportunities per unit (applies only the DPMO chart)

dpmo = defects per million opportunities (applies only the DPMO chart)
calculated as: dpmo = (1,000,000 * numberOfDefects) / (sampleSize * dopu)

up = estimate (or average) of the dpmo values

Fraction Defective Parts – Also known as Fraction Non-Conforming or p-chart

UCL	=	p + 3 * Sqrt (p * (1- p) / n)
Center line	=	p
LCL	=	p - 3 * Sqrt (p * (1- p) / n)

Percent Defective Parts – Also known as Percent Non-Conforming or p-chart

UCL	=	P + 3 * Sqrt (P * (100% - P) / n)
Center line	=	P
LCL	=	P - 3 * Sqrt (P * (100% - P) / n)

Number of Defective Parts – Also known as the Number Nonconforming or np-chart

UCL	=	(n * p) + 3 * Sqrt ((n * p) * (1- p) / n)
Center line	=	(n * p)
LCL	=	(n * p) - 3 * Sqrt ((n * p) * (1- p) / n)

In this case the value (n * p) represents the average number of defective parts per sample subgroup. Since p is the estimate (or average) of the fraction defective per sample subgroup, n * p is the average number of defective per sample subgroup. Or you can add up all the number defective parts in all subgroups and divide by the number of subgroups, that to will reduce to the average number of defective per sample subgroup

Number of Defects Control Chart – Also known as Number Nonconformities or c-chart

UCL	=	c + 3 * Sqrt (c)
Center line	=	c
LCL	=	c - 3 * Sqrt (c)

Number of Defects per Unit Control Chart – Also known as Number Nonconformities per Unit or u-chart

UCL = u + 3 * Sqrt (u / n)

Center line = u

LCL = u - 3 * Sqrt (u / n)

Number Defects Per Million – Also known as DPMO

UCL = up + 3000 * Sqrt (up /(dopu * n))

Center line = up

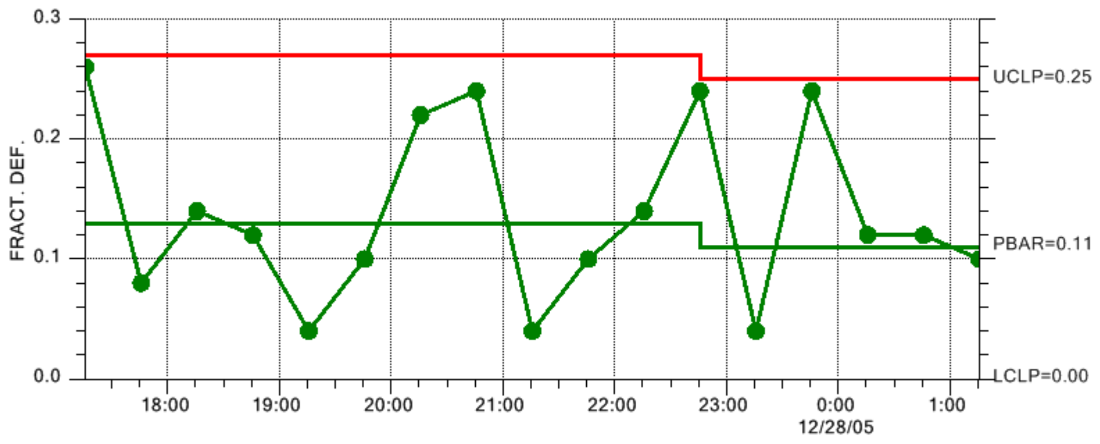
LCL = up - 3000 * Sqrt (up/(dopu * n))

Variable SPC Control Limits

There can be situations where the SPC control limit changes in a chart. If your control limits change, you need to set the following **ControlLineMode** property to SPCChartObjects.CONTROL_LINE_VARIABLE, as in the example below. The default value is SPCChartObjects.CONTROL_LINE_FIXED.

```
this.getPrimaryChart().setControlLineMode(SPCChartObjects.CONTROL_LINE_VARIABLE);
```

In the SPCChartObjects.CONTROL_LINE_FIXED case, the current SPC control limit plots as a horizontal straight line for the entire width of the chart, regardless if the control limit changes, either explicity, or using the **autoCalculateControlLimits** method. If the **ControlLineMode** property is SPCChartObjects.CONTROL_LINE_VARIABLE, the SPC limit value plots at the value it had when the sample subgroup values updated. If you change a control limit value, the control limit line will no longer be a straight horizontal line, instead it will be jagged, or stepped, depending on the changes made.



There are three ways to enter new SPC limit values. See the example program TimeAttributeControlCharts.VariableControlLimits for an example of all three methods. First, you can use the method **getChartData().setControlLimitValues** method.

```
double [] initialControlLimits = {0.13, 0.0, 0.27};
double [] changeControlLimits = {0.11, 0.0, 0.25};
.
.
this.getPrimaryChart().setControlLineMode(SPCChartObjects.CONTROL_LINE_VARIABLE);
.
.
// Change limits at sample subgroup 10
if (i== 10)
{
    this.getChartData().setControlLimitValues (changeControlLimits);
}
this.getChartData().addNewSampleRecord (timestamp, samples);
```

Second, you can use the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

```
.
.
this.getPrimaryChart().setControlLineMode(SPCChartObjects.CONTROL_LINE_VARIABLE);
.
.
```

```
// Variable Control Limits re-calculated every update after 10 using
//  autoCalculateControlLimits
if (i > 10)
    this.autoCalculateControlLimits ();
this.getChartData().addNewSampleRecord (timestamp, samples);
```

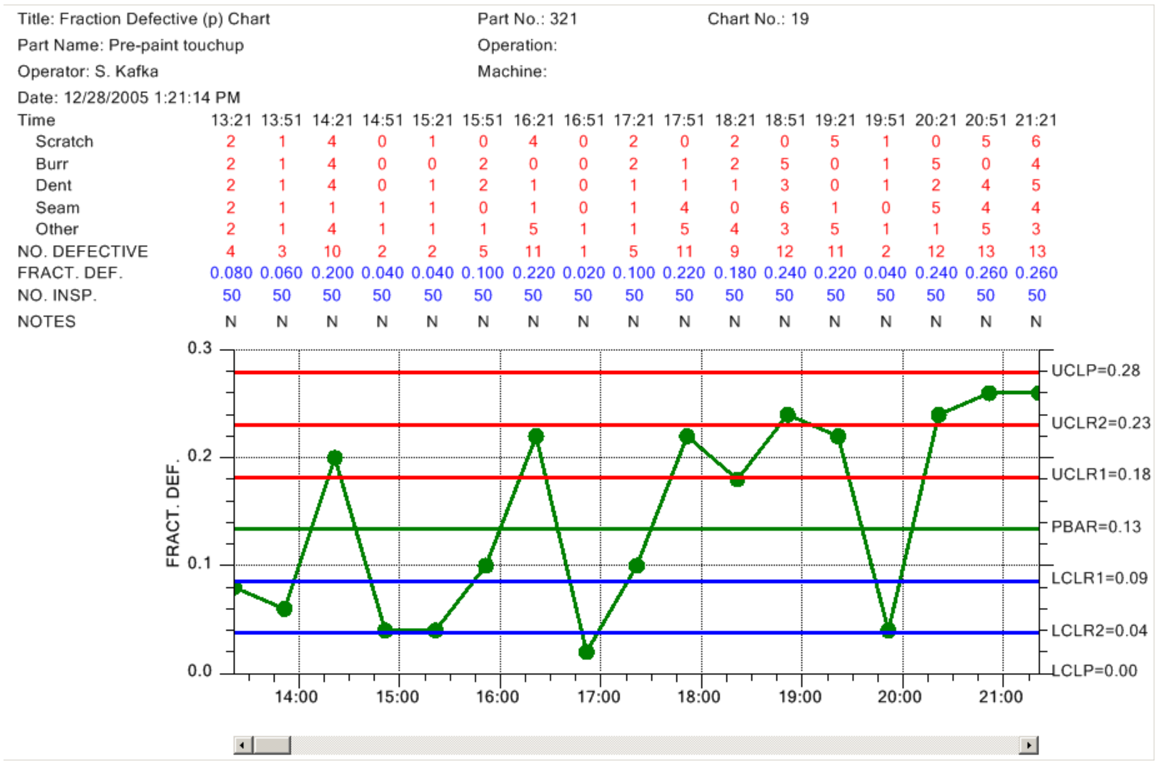
Last, you can enter the SPC control limits with every new sample subgroup record, using one of the methods that include a control limits array parameter.

```
double [] initialControlLimits = {0.13, 0.0, 0.27};
double [] changeControlLimits = {0.11, 0.0, 0.25};
DoubleArray variableControlLimits;.
.
.
this.getPrimaryChart().setControlLineMode(SPCChartObjects.CONTROL_LINE_VARIABLE);
.
.
//      Variable Control Limits updated using addNewSampleRecord
    if (i== 10) // need to convert changeControlLimits to a DoubleArray
        variableControlLimits = new DoubleArray(changeControlLimits);
    this.getChartData().addNewSampleRecord (timestamp, samples,
        variableControlLimits, notesstring);
```

Multiple SPC Control Limits

The normal SPC control limit displays at the 3-sigma level, both high and low. A common standard is that if the process variable under observation falls outside of the +-3-sigma limits the process is out of control. The default setup of our variable control charts have a high limit at the +3-sigma level, a low limit at the -3-sigma level, and a target value. There are situations where the quality engineer also wants to display control limits at the 1-sigma and 2-sigma level. The operator might receive some sort of preliminary warning if the process variable exceeds a 2-sigma limit.
.

You are able to add additional control limit lines to an attribute control chart, as in the example program TimeAttributeControlCharts.MultipleControlLimitsChart.

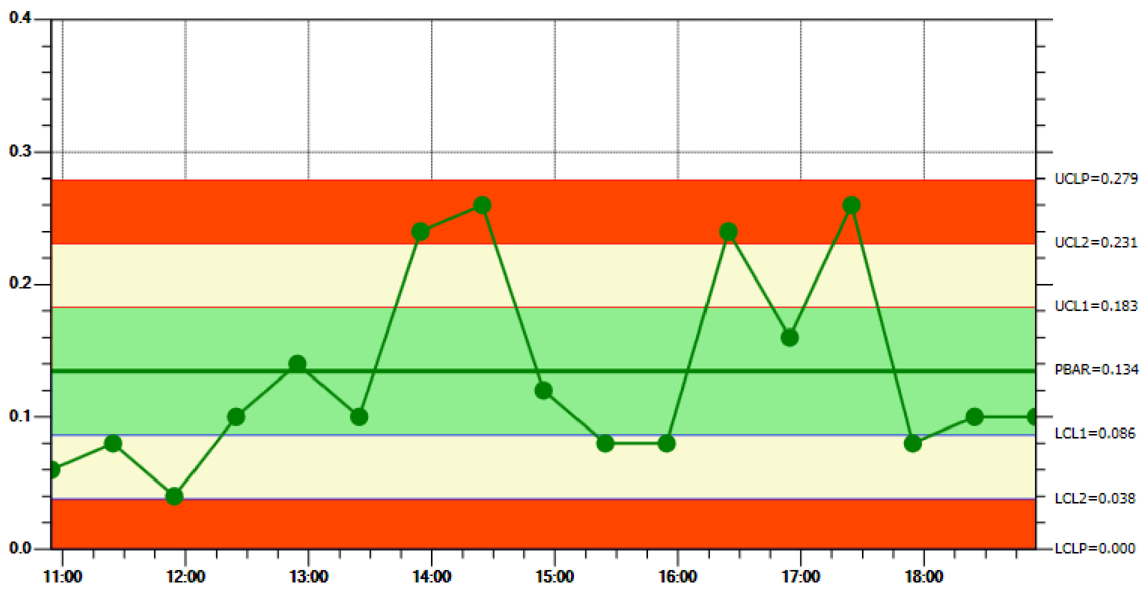


Title: Fraction Defective (p) Chart Part No.: 321 Chart No.: 19
Part Name: Pre-paint touchup Operation:
Operator: S. Kafka Machine:
Date: 12/28/2005 1:21:14 PM

Time	13:21	13:51	14:21	14:51	15:21	15:51	16:21	16:51	17:21	17:51	18:21	18:51	19:21	19:51	20:21	20:51	21:21
Scratch	2	1	4	0	1	0	4	0	2	0	2	0	5	1	0	5	6
Burr	2	1	4	0	0	2	0	0	2	1	2	5	0	1	5	0	4
Dent	2	1	4	0	1	2	1	0	1	1	1	3	0	1	2	4	5
Seam	2	1	1	1	1	0	1	0	1	4	0	6	1	0	5	4	4
Other	2	1	4	1	1	1	5	1	1	5	4	3	5	1	1	5	3
NO. DEFECTIVE	4	3	10	2	2	5	11	1	5	11	9	12	11	2	12	13	13
FRACT. DEF.	0.080	0.060	0.200	0.040	0.040	0.100	0.220	0.020	0.100	0.220	0.180	0.240	0.220	0.040	0.240	0.260	0.260
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

We added a method (**add3SigmaControlLimits**) which will generate multiple control limits, for +-1, 2, and 3- sigma levels, based on an initial specification of the target value, and the +-3 sigma control limits. This is most useful if you want to generate +-1, 2 and 3-sigma control limits in order to fill in between them with a zone fill color. See the TimeAttributeControlCharts.MultiControlLimitsChart example. If you call the **autoCalculateControlLimits** method, the initial +-1,2 and 3-sigma control limit values will be altered to the new, calculated values, but the control limit lines remain, with their new values. Since you do not normally want to be generating alarm messages for excursions into the +-1 and 2-sigma limit areas, the **add3SigmaControl** limits has the option of disabling alarm notification in the case of +-1 and +-2 alarm conditions.

```
 // initial limits, replaced with auto-calculated values
  double ll = 10, hh = 30, target = 20;
  // limitcheck = false means to only check alarms at +-3 sigma limits, not other
limits, even though they are displayed.
 boolean limitcheck = false;
 this.getPrimaryChart().add3SigmaControlLimits(target, ll, hh, limitcheck);
 this.getPrimaryChart().ControlLimitLineFillMode = true;
```



Control Limit Fill Option used with +-1, 2 and 3-sigma control limits

You can also add additional control limits one at a time. By default you get the +-3-sigma control limits. So additional control limits should be considered +-2-sigma and +-1-sigma control limits. Do not confuse control limits with specification limits, which must be added using the **AddSpecLimit** method. There are two steps to adding additional control limits: creating a **SPCControlLimitRecord** object for the new control limit, and adding the control limit to the chart using the charts **AddAdditionalControlLimit** method**.** It is critical that you add them in a specific order, that order being:

Primary Chart	SPC_LOWER_CONTROL_LIMIT_2	(2-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_2	(2-sigma upper limit)
Primary Chart	SPC_LOWER_CONTROL_LIMIT_1	(1-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_1	(1-sigma upper limit)

```
double sigma2 = 2.0;
double sigma1 = 1.0;
// Create multiple limits
SPCControlLimitRecord lcl2 = new SPCControlLimitRecord(this.getChartData(),
    SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0,"LCL2", "LCL2");
SPCControlLimitRecord ucl2 = new SPCControlLimitRecord(this.getChartData(),
    SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 0,"UCL2", "UCL2");

this.getPrimaryChart().addAdditionalControlLimit(lcl2,
    SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_2, sigma2);
this.getPrimaryChart().addAdditionalControlLimit(ucl2,
    SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);
```

```
SPCControlLimitRecord  lcl3 = new SPCControlLimitRecord(this.getChartData(),
    SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 5,"LCL1", "LCL1");
SPCControlLimitRecord  ucl3 = new SPCControlLimitRecord(this.getChartData(),
    SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 35,"UCL1", "UCL1");

this.getPrimaryChart().addAdditionalControlLimit(lcl3,
    SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_1, sigma1);
this.getPrimaryChart().addAdditionalControlLimit(ucl3,
     SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);
```

Special Note – When you create a **SPCControlLimitRecord** object, you can specify an actual limit level. If you do not call the charts **autoCalculateControlLimits** method, the control limit will be displayed at that value. If you do call **autoCalculateControlLimits** method, the auto-calculated value overrides the initial value (0.0 in the examples above). When you call the charts **addAdditionalControlLimits** method, you specify the sigma level that is used by the **autoCalculateControlLimits** to calculate the control limit level.

If you want the control limits displayed as filled areas, set the charts **ControlLimitLineFillMode** property True.

```
      this.getPrimaryChart().setControlLimitLineFillMode(true);
```

This will fill each control limit line from the limit line to the target value of the chart. In order for the fill to work properly, you must set this property after you define all additional control limits. In order for the algorithm to work, you must add the outer most control limits (SPC_UPPER_CONTROL_LIMIT_3 and SPC_LOWER_CONTROL_LIMIT_3) first, followed by the next outer most limits (SPC_UPPER_CONTROL_LIMIT_2 and SPC_LOWER_CONTROL_LIMIT_2), followed by the inner most control limits (SPC_UPPER_CONTROL_LIMIT_1 and SPC_LOWER_CONTROL_LIMIT_1). This way the fill of the inner limits will partially cover the fill of the outer limits, creating the familiar striped look you want to see.

Chart Y-Scale

You can set the minimum and maximum values of the two charts y-scales manually using the **getPrimaryChart().setMinY**, **getPrimaryChart().setMaxY**, **getSecondaryChart().setMinY** and **getSecondaryChart().setMaxY** methods.

```
// Set initial scale of the y-axis of the mean chart
// If you are calling AutoScalePrimaryChartYRange this isn't really needed
this.getPrimaryChart().setMinY( 0);
this.getPrimaryChart().setMaxY( 40);
```

It is easiest to just call the auto-scale routines after the chart has been initialized with data, and any control limits calculated.

```
// Must have data loaded before any of the Auto.. methods are called
    simulateData();

// Calculate the SPC control limits for both graphs of the current SPC chart
    this.autoCalculateControlLimits ();

// Scale the y-axis of the X-Bar chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
```

Once all of the graph parameters are set, call the method **rebuildChartUsingCurrentData** .

```
// Rebuild the chart using the current data and settings
this.rebuildChartUsingCurrentData ();
```

If, at any future time you change any of the chart properties, you will need to call **rebuildChartUsingCurrentData** to force a rebuild of the chart, taking into account the current properties. **rebuildChartUsingCurrentData** invalidates the chart and forces a redraw**.** Our examples that update dynamically demonstrate this technique. The chart is setup with some initial settings and data values. As data is added in real-time to the graph, the chart SPC limits, and y-scales are constantly recalculated to take into account new data values. The code below is extracted from the TimeAttributeControlCharts.DynamicAttributeControlChart example program.

```
private void timer1_Tick()
{
    GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
    // This simulates an assignable defect for each category, the last category
    //  is assigned the total number of defective parts, not defects.
    DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
SPCControlChartData.PERCENT_DEFECTIVE_PARTS_CHART);
    // Add new sample record
    this.getChartData().addNewSampleRecord(timestamp, samples);
    // Simulate 30  minute passing
    startTime.add(ChartObj.MINUTE, 30);

    // Calculate the SPC control limits
    this.autoCalculatePrimaryControlLimits();
    // Scale the y-axis of the SPC chart to display all data and control limits
    this.autoScalePrimaryChartYRange();
    // Rebuild the chart using the current data and settings
    this.rebuildChartUsingCurrentData();
    this.updateDraw();
}
```

Updating Chart Data

The real-time example above demonstrates how the SPC chart data is updated, using the **getChartData().addNewSampleRecord** method. In this case, the chart data updates with each timer tick event, though it could just as easily be any other type of event. If you have already collected all of your data and just want to plot it all at once, use a simple loop like most of our examples do to update the data.

```
private void simulateData()
for (int i=0; i < 200; i++)
{
    GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
    // This simulates an assignable defect for each category, the last category
    //  is assigned the total number of defective parts, not defects.
    DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
SPCControlChartData.PERCENT_DEFECTIVE_PARTS_CHART);
    // Add new sample record
    this.getChartData().addNewSampleRecord(timestamp, samples, notesstring);
    // Simulate 30  minute passing
    startTime.add(ChartObj.MINUTE, 30);
}
}
```

In this example the sample data and the time stamp for each sample record is simulated. In your application, you will probably be reading the sample record values from some sort of database or file, along with the actual time stamp for that data.

If you want to append a text note to a sample record, use one of the **getChartData().addNewSampleRecord** overrides that have a *notes* parameter. The code below is extracted from the TimeAttributeControlCharts.SimpleAttributeControlChart example.

```
private void simulateData()
{  String notesstring = "";
    for (int i=0; i < 200; i++)
    {
        GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
    // This simulates an assignable defect for each category, the last category
    //  is assigned the total number of defective parts, not defects.
        DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
SPCControlChartData.PERCENT_DEFECTIVE_PARTS_CHART);


        double r = ChartSupport.getRandomDouble();
        if (r < 0.1) // make a note on every tenth item, on average
            notesstring = "Note for sample subgroup #" + Integer.toString(i) + ".
Spray paint nozzel clogged. Replaced with new, Enois nozzle.";
        else
            notesstring = "";
        // Add new sample record
        this.getChartData().addNewSampleRecord(timestamp, samples, notesstring);
        // Simulate 30  minute passing
        startTime.add(ChartObj.MINUTE, 30);
    }
}
```

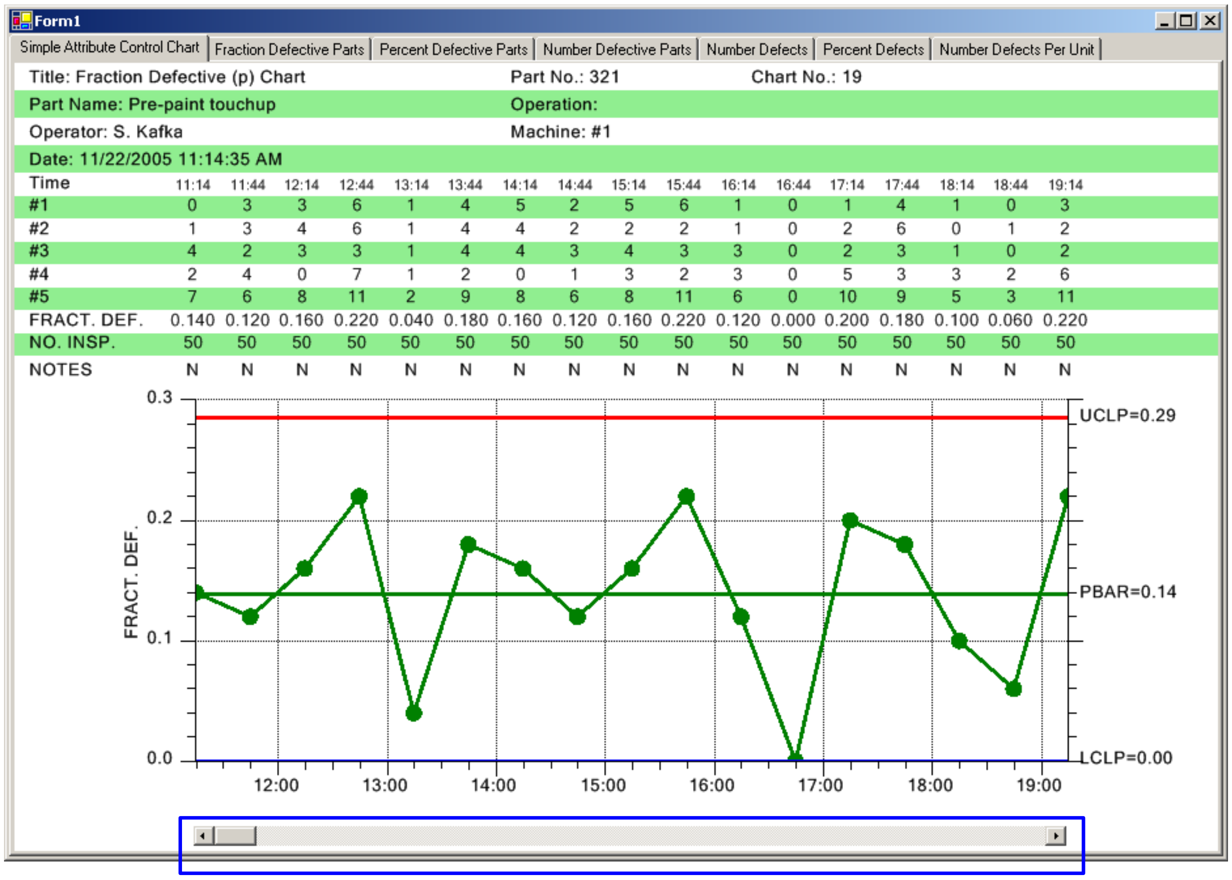
Scatter Plots of the Actual Sampled Data

⑤ This option is not applicable for attribute control charts.

Enable Chart ScrollBar

Set the **EnableScrollBar** property true to enable the chart scrollbar. You will then be able to window in on 8-20 sample subgroups at a time, from a much larger collection of measurement data representing hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.

```
// enable scroll bar
this.setEnableScrollBar( true);
```



Title: Fraction Defective (p) Chart   Part No.: 321   Chart No.: 19

Part Name: Pre-paint touchup   Operation:

Operator: S. Kafka   Machine: #1

Date: 11/22/2005 11:14:35 AM

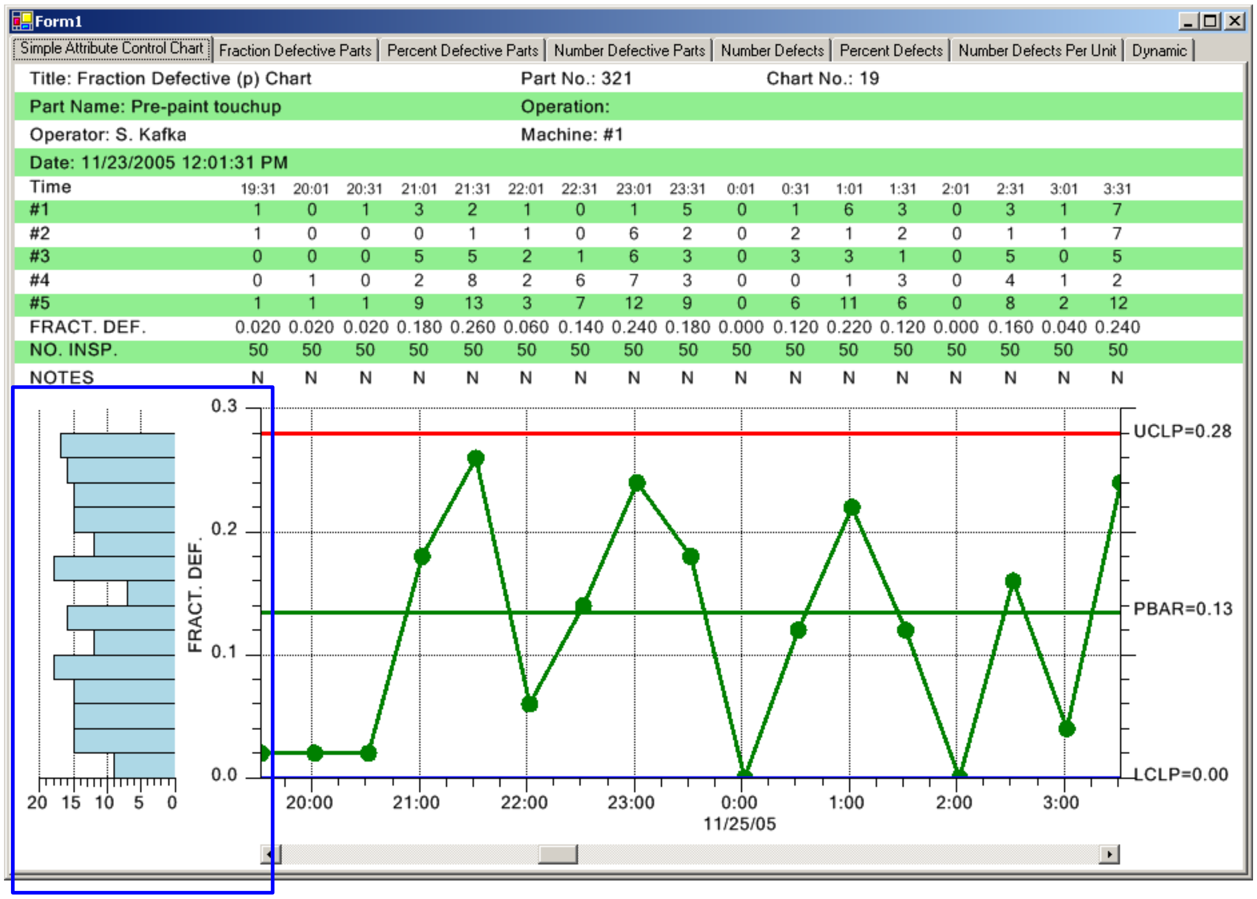
Time	11:14	11:44	12:14	12:44	13:14	13:44	14:14	14:44	15:14	15:44	16:14	16:44	17:14	17:44	18:14	18:44	19:14
#1	0	3	3	6	1	4	5	2	5	6	1	0	1	4	1	0	3
#2	1	3	4	6	1	4	4	2	2	2	1	0	2	6	0	1	2
#3	4	2	3	3	1	4	4	3	4	3	3	0	2	3	1	0	2
#4	2	4	0	7	1	2	0	1	3	2	3	0	5	3	3	2	6
#5	7	6	8	11	2	9	8	6	8	11	6	0	10	9	5	3	11
FRACT. DEF.	0.140	0.120	0.160	0.220	0.040	0.180	0.160	0.120	0.160	0.220	0.120	0.000	0.200	0.180	0.100	0.060	0.220
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Scrollbar

SPC Chart Histograms

Viewing frequency histograms of the variation in the primary variable side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process. You can turn on integrated frequency histograms for either chart using the **getPrimaryChart().DisplayFrequencyHistogram** property of the chart.

```
//  frequency histogram for both charts
this.getPrimaryChart().setDisplayFrequencyHistogram( true);
```



Title: Fraction Defective (p) Chart Part No.: 321 Chart No.: 19

Part Name: Pre-paint touchup Operation:

Operator: S. Kafka Machine: #1

Date: 11/23/2005 12:01:31 PM

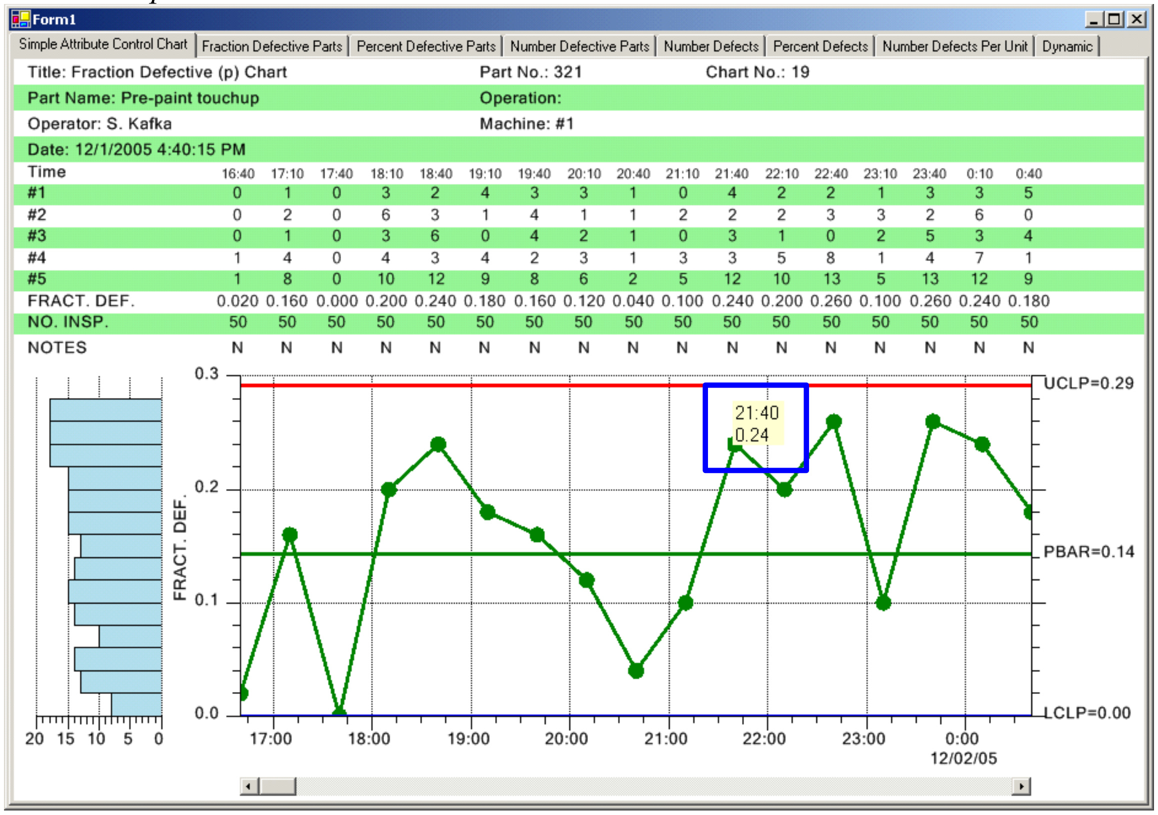
Time	19:31	20:01	20:31	21:01	21:31	22:01	22:31	23:01	23:31	0:01	0:31	1:01	1:31	2:01	2:31	3:01	3:31
#1	1	0	1	3	2	1	0	1	5	0	1	6	3	0	3	1	7
#2	1	0	0	0	1	1	0	6	2	0	2	1	2	0	1	1	7
#3	0	0	0	5	5	2	1	6	3	0	3	3	1	0	5	0	5
#4	0	1	0	2	8	2	6	7	3	0	0	1	3	0	4	1	2
#5	1	1	1	9	13	3	7	12	9	0	6	11	6	0	8	2	12
FRACT. DEF.	0.020	0.020	0.020	0.180	0.260	0.060	0.140	0.240	0.180	0.000	0.120	0.220	0.120	0.000	0.160	0.040	0.240
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Frequency Histogram

SPC Chart Data and Notes Tooltips

You can invoke two types of tooltips using the mouse. The first is a data tooltip. When you hold the mouse button down over one of the data points in the primary chart, the x and y values for that data point display in a popup tooltip..

Data Tooltip

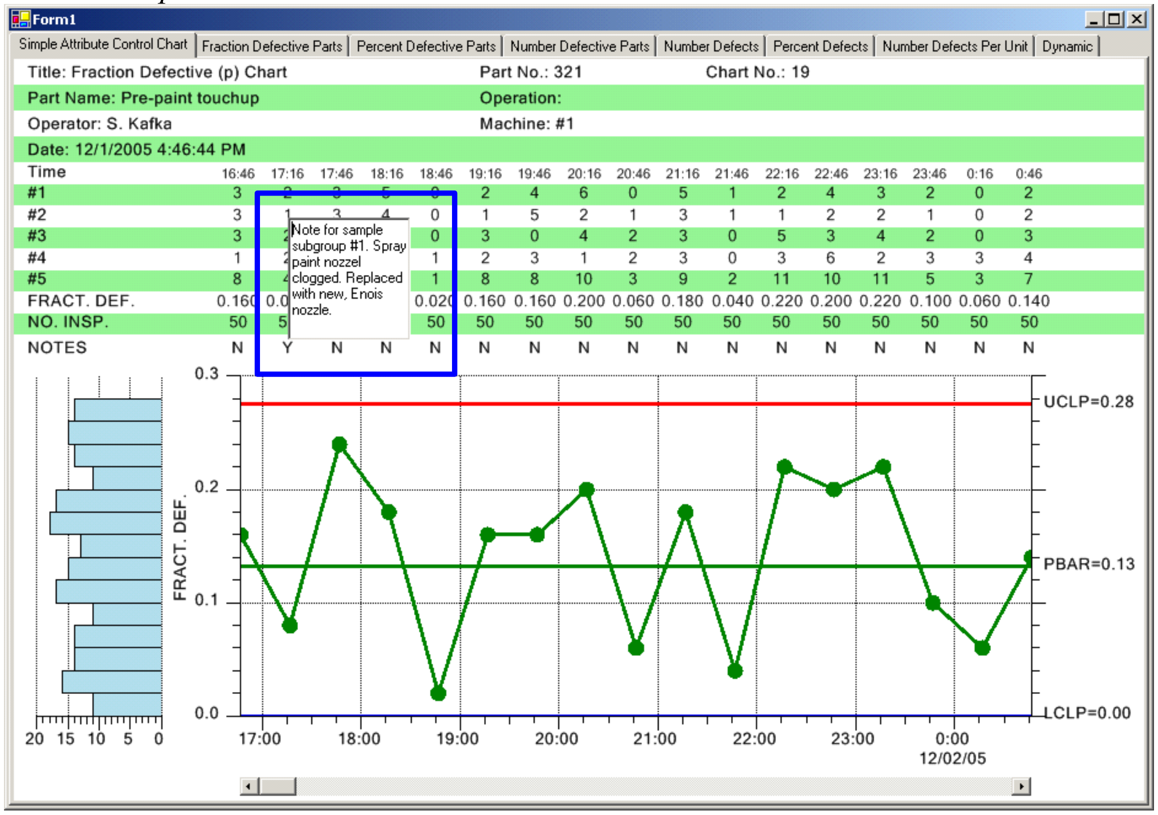


Title: Fraction Defective (p) Chart | Part No.: 321 | Chart No.: 19
Part Name: Pre-paint touchup | Operation:
Operator: S. Kafka | Machine: #1
Date: 12/1/2005 4:40:15 PM

Time	16:40	17:10	17:40	18:10	18:40	19:10	19:40	20:10	20:40	21:10	21:40	22:10	22:40	23:10	23:40	0:10	0:40
#1	0	1	0	3	2	4	3	3	1	0	4	2	2	1	3	3	5
#2	0	2	0	6	3	1	4	1	1	2	2	2	3	3	2	6	0
#3	0	1	0	3	6	0	4	2	1	0	3	1	0	2	5	3	4
#4	1	4	0	4	3	4	2	3	1	3	3	5	8	1	4	7	1
#5	1	8	0	10	12	9	8	6	2	5	12	10	13	5	13	12	9
FRACT. DEF.	0.020	0.160	0.000	0.200	0.240	0.180	0.160	0.120	0.040	0.100	0.240	0.200	0.260	0.100	0.260	0.240	0.180
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

If you are displaying the Notes line in the table portion of the chart, the Notes entry for a sample subgroup displays "Y" if a note was recorded for that sample subgroup, or "N" if no note was recorded. Notes are recorded using one of the **getChartData().addNewSampleRecord** overrides that include a notes parameter. See the section Updating Chart Data. If you click on a "Y" in the Notes row for a sample subgroup, the complete text of the note for that sample subgroup will display in a **JTextArea**, immediately above the "Y". You can actually edit the notes in the **JTextArea**.

Notes Tooltip



Title: Fraction Defective (p) Chart Part No.: 321 Chart No.: 19
Part Name: Pre-paint touchup Operation:
Operator: S. Kafka Machine: #1
Date: 12/1/2005 4:46:44 PM

Time	16:46	17:16	17:46	18:16	18:46	19:16	19:46	20:16	20:46	21:16	21:46	22:16	22:46	23:16	23:46	0:16	0:46
#1	3	2	3	5	0	2	4	6	0	5	1	2	4	3	2	0	2
#2	3	1	3	4	0	1	5	2	1	3	1	1	2	2	1	0	2
#3	3	2			0	3	0	4	2	3	0	5	3	4	2	0	3
#4	1	2			1	2	3	1	2	3	0	3	6	2	3	3	4
#5	8	4			1	8	8	10	3	9	2	11	10	11	5	3	7
FRACT. DEF.	0.160	0.0			0.020	0.160	0.160	0.200	0.060	0.180	0.040	0.220	0.200	0.220	0.100	0.060	0.140
NO. INSP.	50	5			50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Note for sample subgroup #1. Spray paint nozzel clogged. Replaced with new, Enois nozzle.

```
private void simulateData()
{   String notesstring = "";
    for (int i=0; i < 200; i++)
    {
        GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
    // This simulates an assignable defect for each category, the last category
    //  is assigned the total number of defective parts, not defects.
        DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
SPCControlChartData.PERCENT_DEFECTIVE_PARTS_CHART);


        double r = ChartSupport.getRandomDouble();
        if (r < 0.1) // make a note on every tenth item, on average
            notesstring = "Note for sample subgroup #" + Integer.toString(i) + ".
Spray paint nozzel clogged. Replaced with new, Enois nozzle.";
        else
            notesstring = "";


        // Add new sample record
        this.getChartData().addNewSampleRecord(timestamp, samples, notesstring);
        // Simulate 30  minute passing
        startTime.add(ChartObj.MINUTE, 30);
        }
    }
```

Both kinds of tooltips are on by default. Turn the tooltips on or off in the program using the **EnableDataToolTip** and **EnableNotesToolTip** flags.

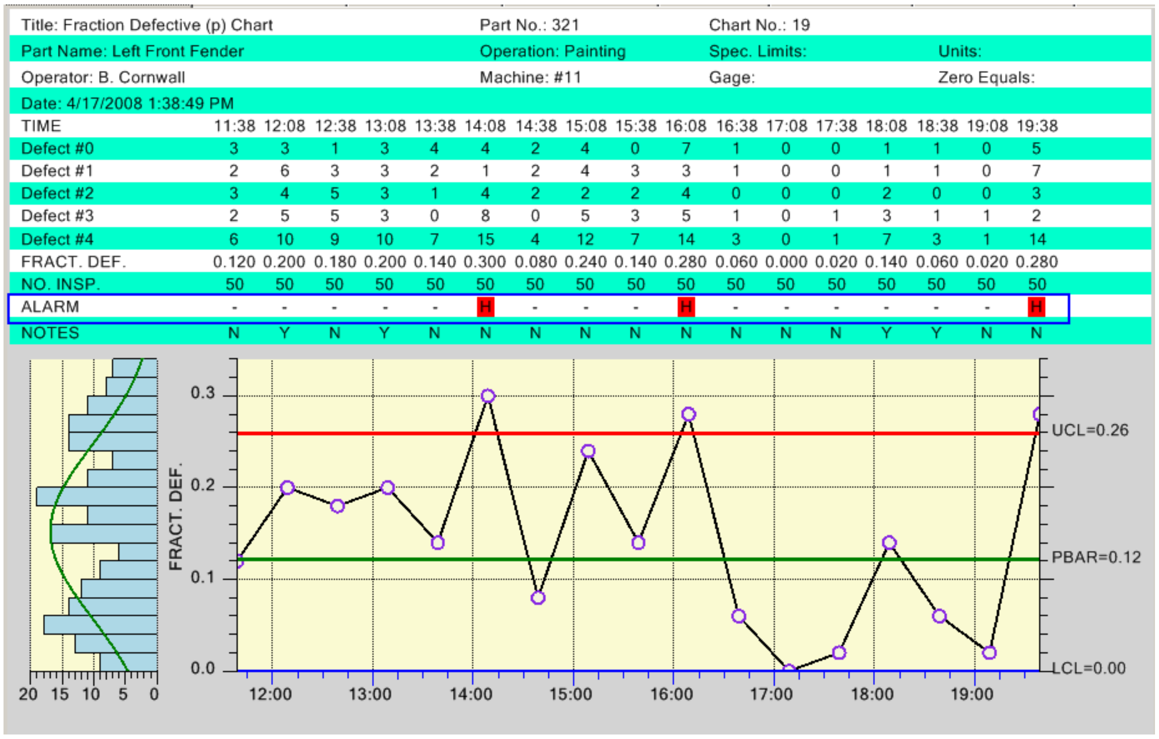
```
// Enable data and notes tooltips
this.setEnableDataToolTip (true);
this.setEnableNotesToolTip( true);
```

The notes tooltip has an additional option. In order to make the notes tooltip "editable", the tooltip, which is Java **JTextArea**, displays on the first click, and goes away on the second click. You can click inside the **JTextArea** and not worry the tooltip suddenly disappearing. The notes tooltip works this way by default. If you wish to explicitly set it, or change it so that the tooltip only displays while the mouse button is held down, as the data tooltips do, set the **getChartData().NotesToolTips.ToolTipMode** property to **NotesToolTip.MOUSEDOWN_TOOLTIP**, as in the example below.

```
// Enable data and notes tooltips
this.setEnableDataToolTip( true);
this.setEnableNotesToolTip(true);

chartdata.getNotesToolTips().setButtonMask ( InputEvent.BUTTON1_MASK);
this.getChartData().getNotesToolTips().setToolTipMode( NotesToolTi
p.MOUSEDOWN_TOOLTIP);
```

Enable Alarm Highlighting

EnableAlarmStatusValues



Title: Fraction Defective (p) Chart Part No.: 321 Chart No.: 19

Part Name: Left Front Fender Operation: Painting Spec. Limits: Units:

Operator: B. Cornwall Machine: #11 Gage: Zero Equals:

Date: 4/17/2008 1:38:49 PM

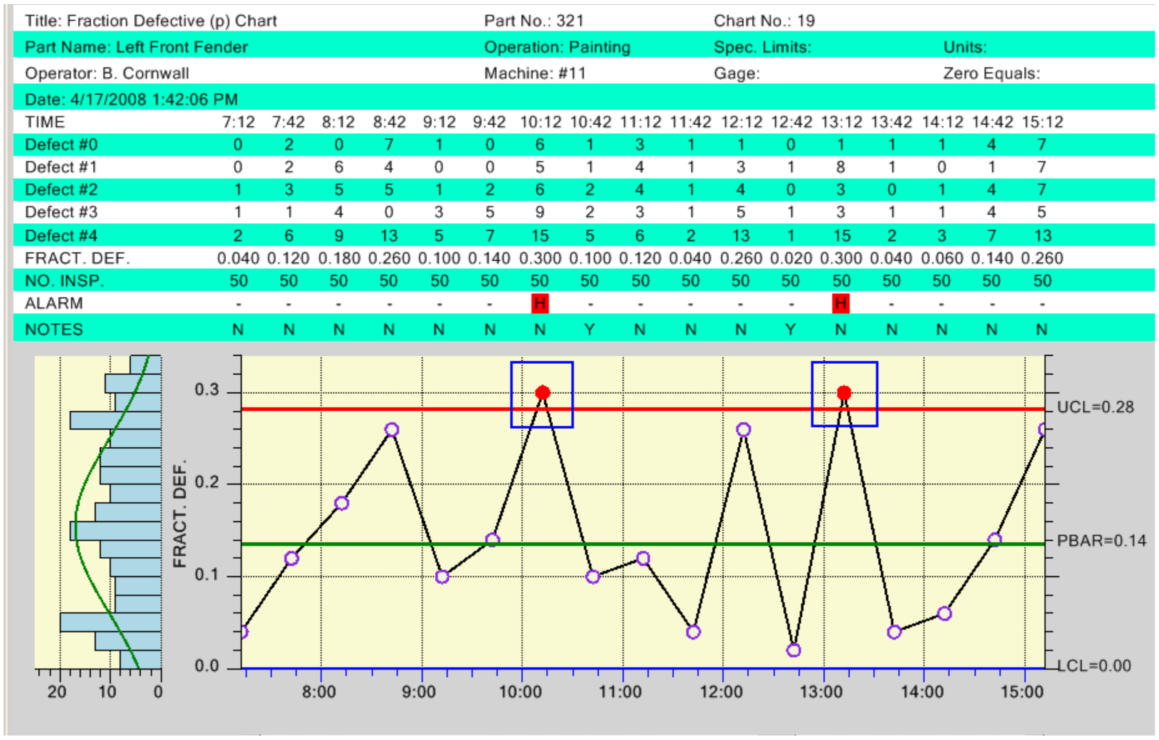
TIME	11:38	12:08	12:38	13:08	13:38	14:08	14:38	15:08	15:38	16:08	16:38	17:08	17:38	18:08	18:38	19:08	19:38
Defect #0	3	3	1	3	4	4	2	4	0	7	1	0	0	1	1	0	5
Defect #1	2	6	3	3	2	1	2	4	3	3	1	0	0	1	1	0	7
Defect #2	3	4	5	3	1	4	2	2	2	4	0	0	0	2	0	0	3
Defect #3	2	5	5	3	0	8	0	5	3	5	1	0	1	3	1	1	2
Defect #4	6	10	9	10	7	15	4	12	7	14	3	0	1	7	3	1	14
FRACT. DEF.	0.120	0.200	0.180	0.200	0.140	0.300	0.080	0.240	0.140	0.280	0.060	0.000	0.020	0.140	0.060	0.020	0.280
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	H	-	-	-	H	-	-	-	-	-	-	H
NOTES	N	Y	N	Y	N	N	N	N	N	N	N	N	N	Y	Y	N	N

There are several alarm highlighting options you can turn on and off. The alarm status line above is turned on/off using the EnableAlarmStatusValues property. We have set it on by default, so you will have to turn it off if you don't want it. Each sample interval has two small boxes that are labeled using one of several different characters, listed below. The most common are an "H" signifying a high alarm, a "L" signifying a low alarm, and a "-" signifying that there is no alarm. When specialized control rules are implemented, either using the named rules discussed in Chapter 8, or custom rules involving trending, oscillation, or stratification, a "T", "O" or "S" may also appear.

"-"	No alarm condition
"H"	High - Measured value is above a high limit
"L"	Low - Measured value falls below a low limit
"T"	Trending - Measured value is trending up (or down).
"O"	Oscillation - Measured value is oscillating (alternating) up and down.
"S"	Stratification - Measured value is stuck in a narrow band.

```
// Alarm status line
this.setEnableAlarmStatusValues(false);
```

ChartAlarmEmphasisMode



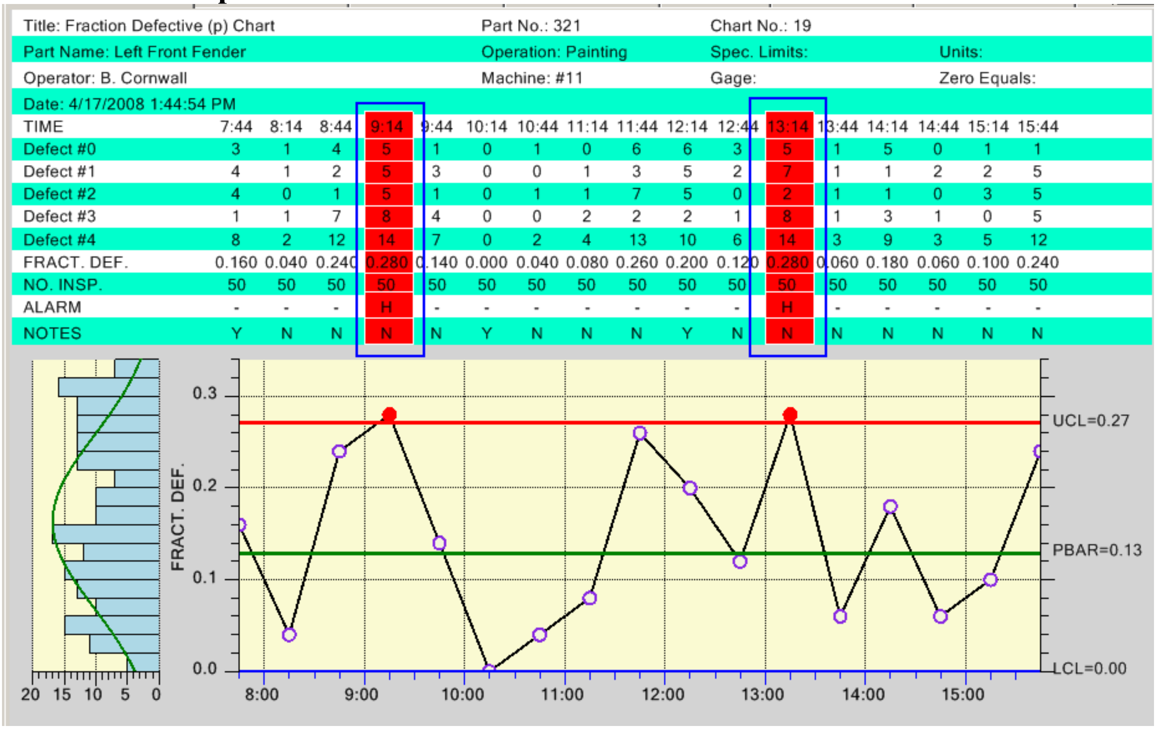
Title: Fraction Defective (p) Chart	Part No.: 321	Chart No.: 19	
Part Name: Left Front Fender	Operation: Painting	Spec. Limits:	Units:
Operator: B. Cornwall	Machine: #11	Gage:	Zero Equals:
Date: 4/17/2008 1:42:06 PM			

TIME	7:12	7:42	8:12	8:42	9:12	9:42	10:12	10:42	11:12	11:42	12:12	12:42	13:12	13:42	14:12	14:42	15:12
Defect #0	0	2	0	7	1	0	6	1	3	1	1	0	1	1	1	4	7
Defect #1	0	2	6	4	0	0	5	1	4	1	3	1	8	1	0	1	7
Defect #2	1	3	5	5	1	2	6	2	4	1	4	0	3	0	1	4	7
Defect #3	1	1	4	0	3	5	9	2	3	1	5	1	3	1	1	4	5
Defect #4	2	6	9	13	5	7	15	5	6	2	13	1	15	2	3	7	13
FRACT. DEF.	0.040	0.120	0.180	0.260	0.100	0.140	0.300	0.100	0.120	0.040	0.260	0.020	0.300	0.040	0.060	0.140	0.260
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	-	H	-	-	-	-	-	H	-	-	-	-
NOTES	N	N	N	N	N	N	N	Y	N	N	N	Y	N	N	N	N	N

```
// Chart alarm emphasis mode
this.setChartAlarmEmphasisMode( SPCChartBase.ALARM_HIGHLIGHT_SYMBOL);
```

The scatter plot symbol used to plot a data point in the chart is normally a fixed color circle. If you turn on the alarm highlighting for chart symbols the symbol color for a sample interval that is in an alarm condition will change to reflect the color of the associated alarm line. In the example above, a low alarm (blue circle) occurs at the beginning of the chart and a high alarm (red circle) occurs at the end of the chart. Alarm symbol highlighting is turned on by default. To turn it off use the SPCChartBase.ALARM_NO_HIGHLIGHT_SYMBOL constants.

TableAlarmEmphasisMode -



Title: Fraction Defective (p) Chart	Part No.: 321	Chart No.: 19	
Part Name: Left Front Fender	Operation: Painting	Spec. Limits:	Units:
Operator: B. Cornwall	Machine: #11	Gage:	Zero Equals:
Date: 4/17/2008 1:44:54 PM			

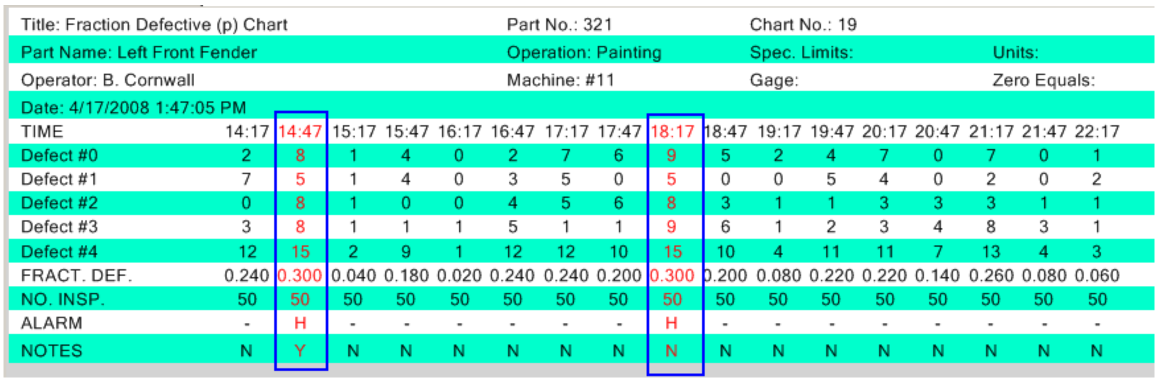
TIME	7:44	8:14	8:44	9:14	9:44	10:14	10:44	11:14	11:44	12:14	12:44	13:14	13:44	14:14	14:44	15:14	15:44
Defect #0	3	1	4	5	1	0	1	0	6	6	3	5	1	5	0	1	1
Defect #1	4	1	2	5	3	0	0	1	3	5	2	7	1	1	2	2	5
Defect #2	4	0	1	5	1	0	1	1	7	5	0	2	1	1	0	3	5
Defect #3	1	1	7	8	4	0	0	2	2	2	1	8	1	3	1	0	5
Defect #4	8	2	12	14	7	0	2	4	13	10	6	14	3	9	3	5	12
FRACT. DEF.	0.160	0.040	0.240	0.280	0.140	0.000	0.040	0.080	0.260	0.200	0.120	0.280	0.060	0.180	0.060	0.100	0.240
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	H	-	-	-	-	-	-	-	H	-	-	-	-	-
NOTES	Y	N	N	N	N	Y	N	N	N	Y	N	N	N	N	N	N	N

```
// Table alarm emphasis mode
this.setTableAlarmEmphasisMode (SPCChartBase.ALARM_HIGHLIGHT_BAR);
```

The entire column of the data table can be highlighted when an alarm occurs. There are four modes associated with this property:

ALARM_HIGHLIGHT_NONE	No alarm highlight
ALARM_HIGHLIGHT_TEXT	Text alarm highlight
ALARM_HIGHLIGHT_OUTLINE	Outline alarm highlight
ALARM_HIGHLIGHT_BAR	Bar alarm highlight

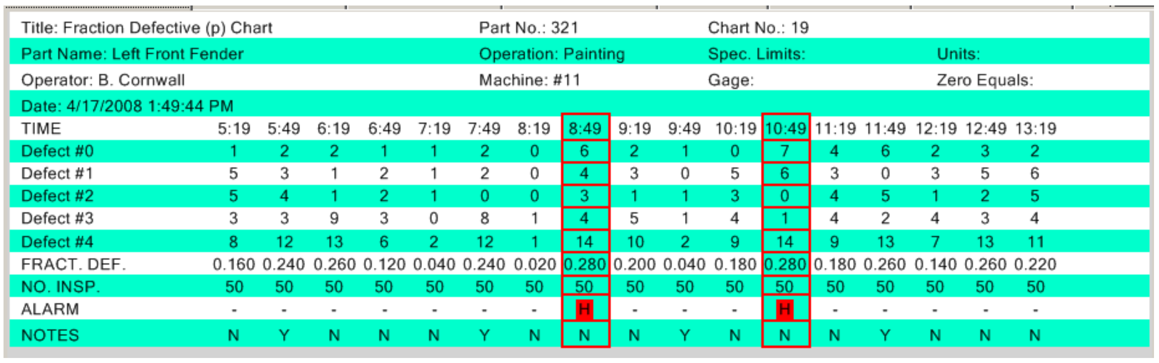
The example above uses the ALARM_HIGHLIGHT_BAR mode.



Title: Fraction Defective (p) Chart	Part No.: 321	Chart No.: 19	
Part Name: Left Front Fender	Operation: Painting	Spec. Limits:	Units:
Operator: B. Cornwall	Machine: #11	Gage:	Zero Equals:
Date: 4/17/2008 1:47:05 PM			

TIME	14:17	14:47	15:17	15:47	16:17	16:47	17:17	17:47	18:17	18:47	19:17	19:47	20:17	20:47	21:17	21:47	22:17
Defect #0	2	8	1	4	0	2	7	6	9	5	2	4	7	0	7	0	1
Defect #1	7	5	1	4	0	3	5	0	5	0	0	5	4	0	2	0	2
Defect #2	0	8	1	0	0	4	5	6	8	3	1	1	3	3	3	1	1
Defect #3	3	8	1	1	1	5	1	1	9	6	1	2	3	4	8	3	1
Defect #4	12	15	2	9	1	12	12	10	15	10	4	11	11	7	13	4	3
FRACT. DEF.	0.240	0.300	0.040	0.180	0.020	0.240	0.240	0.200	0.300	0.200	0.080	0.220	0.220	0.140	0.260	0.080	0.060
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	H	-	-	-	-	-	-	H	-	-	-	-	-	-	-	-
NOTES	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_TEXT mode



Title: Fraction Defective (p) Chart | Part No.: 321 | Chart No.: 19

Part Name: Left Front Fender | Operation: Painting | Spec. Limits: | Units:

Operator: B. Cornwall | Machine: #11 | Gage: | Zero Equals:

Date: 4/17/2008 1:49:44 PM

TIME	5:19	5:49	6:19	6:49	7:19	7:49	8:19	8:49	9:19	9:49	10:19	10:49	11:19	11:49	12:19	12:49	13:19
Defect #0	1	2	2	1	1	2	0	6	2	1	0	7	4	6	2	3	2
Defect #1	5	3	1	2	1	2	0	4	3	0	5	6	3	0	3	5	6
Defect #2	5	4	1	2	1	0	0	3	1	1	3	0	4	5	1	2	5
Defect #3	3	3	9	3	0	8	1	4	5	1	4	1	4	2	4	3	4
Defect #4	8	12	13	6	2	12	1	14	10	2	9	14	9	13	7	13	11
FRACT. DEF.	0.160	0.240	0.260	0.120	0.040	0.240	0.020	0.280	0.200	0.040	0.180	0.280	0.180	0.260	0.140	0.260	0.220
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	-	-	H	-	-	-	H	-	-	-	-	-
NOTES	N	Y	N	N	N	Y	N	N	N	Y	N	N	N	Y	N	N	N

The example above uses the ALARM_HIGHLIGHT_OUTLINE mode. In the table above, the column outlines in blue and red reflect what is actually displayed in the chart, whereas in the other TableAlarmEmphasisMode examples the outline just shows where the alarm highlighting occurs.

The default mode is ALARM_HIGHLIGHT_NONE mode.

AutoLogAlarmsAsNotes

When an alarm occurs, details of the alarm can be automatically logged as a Notes record. Just set the setAutoLogAlarmsAsNotes method to true.

```
this.setAutoLogAlarmsAsNotes(true);
```

Creating a Batch-Based Attribute Control Chart

Both the **SPCTimeAttributeContolChart** and **SPCBatchAttributeControlChart** derive from the **SPCChartBase** and as a result the two classes are very similar and share 95% of all properties in common. Creating and initializing a batch-based SPC chart is much the same as that of a time-based SPC chart. Derive your base class from **SPCBatchAttributeControlChart** class as seen in the BatchAttributeControlChart.SimpleAttributeControlChart example program.

```
public class SimpleAttributeControlChart extends SPCBatchAttributeControlChart {


    GregorianCalendar startTime = new GregorianCalendar();

    int batchCounter = 0;


    //  SPC attribute control chart type

    int charttype = SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
    // Number of samples per sub group
    int numsamplespersubgroup = 50;
    // Number of defect categories
    int numcategories = 5;
    // Number of datapoints in the view
    int numdatapointsinview = 17;
    // The time increment between adjacent subgroups
    int timeincrementminutes = 30;


    public SimpleAttributeControlChart()
    {

        // Define and draw chart
        initializeChart();
    }


    public void initializeChart()
    {
        // Initialize the SPCTimeAttributeControlChart
        this.initSPCBatchAttributeControlChart(charttype, numcategories,
            numsamplespersubgroup, numdatapointsinview);

.
.
.

    }
```

Establish the control chart type (p-, np-, c- or u-chart) using the attribute control charts **initSPCBatchAttributeControlChart** initialization routine.

SPCBatchAttributeControlChart.initSPCBatchAttributeControlChart Method

This initialization method initializes the most important values in the creation of a SPC chart.

public void initSPCBatchAttributeControlChart(
  int *charttype*,
  int *numcategories*,
  int *numsamplespersubgroup*,

  int *numdatapointsinview*,
);

Parameters

charttype
: Specifies the chart type. Use one of the SPC Attribute Control chart types: PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART.

numcategories
: In Attribute Control Charts this value represents the number of defect categories used to determine defect counts.

numsamplespersubgroup
: In an Attribute Control chart it represents the total sample size per sample subgroup from which the defect data is counted.

numdatapointsinview
: Specifies the number of sample subgroups displayed in the graph at one time.

Update the chart data using the **getChartData().addNewSampleRecord** method, using an override that has the batch number (*batchCounter* below) as the first parameter. Even though a time stamp value is used in the **addNewSampleRecord** method, it is not used in the actual graph. Instead, it is used as the time stamp for the batch in the table portion of the chart.

```
    private void simulateData()
    {
        for (int i=0; i < 200; i++)
        {
        double batchnumber = batchCounter;
        GregorianCalendar timestamp = (GregorianCalendar) startTime.clone();
        // Simulate a new sample record
        DoubleArray samples = this.getChartData().simulateDefectRecord(50 * 0.134,
SPCControlChartData.PERCENT_DEFECTIVE_PARTS_CHART);
        //  add a new sample record
        this.getChartData().addNewSampleRecord(batchnumber, timestamp, samples);
        // Simulate timeincrementminutes  minute passing
        startTime.add(ChartObj.MINUTE, timeincrementminutes);
        batchCounter++;
        }
    }
```

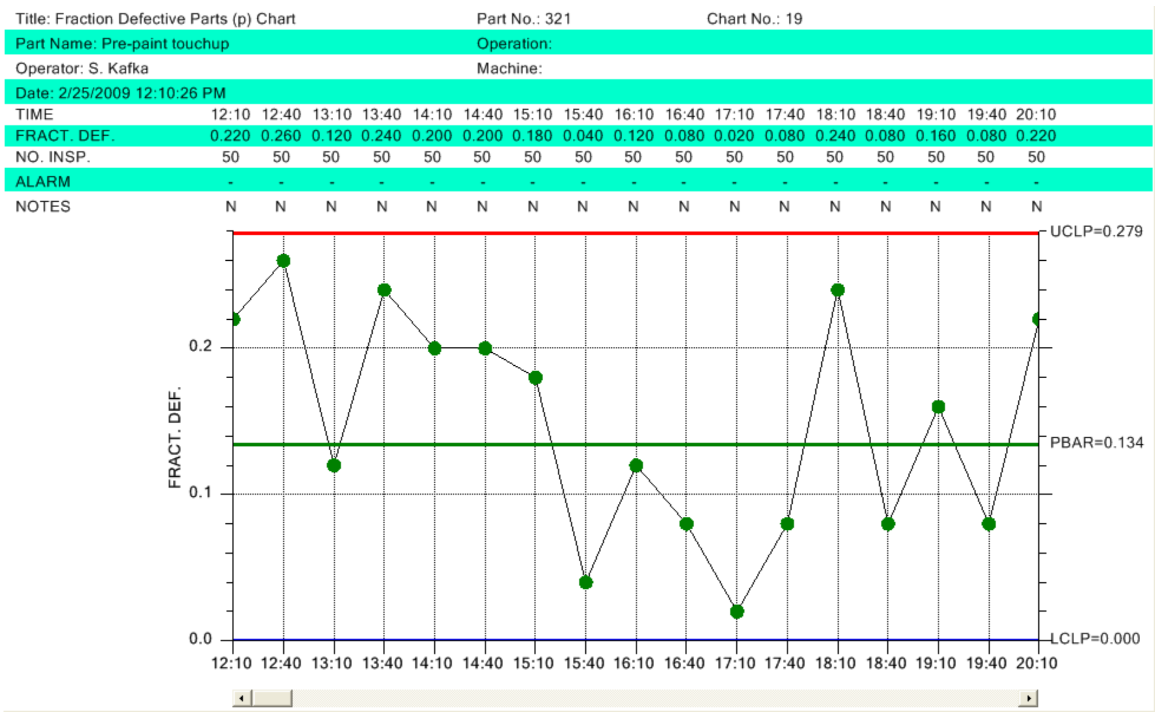
Changing the Batch Control Chart X-Axis Labeling Mode

In revisions prior to 2.0, the x-axis tick marks of a batch control chart could only be labeled with the numeric batch number of the sample subgroup. While batch number labeling is still the default mode, it is now possible to label the sample subgroup tick marks using the time stamp of the sample subgroup, or a user-defined string unique to each sample subgroup.

You may find that labeling every subgroup tick mark with a time stamp, or a user-defined string, causes the axis labels to stagger because there is not enough room to display the tick mark label without overlapping its neighbor. In these cases you may wish to reduce the number of sample subgroups you show on the page using the *`numdatapointsinview`* `variable` found in all of the example programs.

```
// Number of datapoints in the view
int numdatapointsinview = 13;
```

Batch Control Chart X-Axis Time Stamp Labeling



Title: Fraction Defective Parts (p) Chart   Part No.: 321   Chart No.: 19

Part Name: Pre-paint touchup   Operation:

Operator: S. Kafka   Machine:

Date: 2/25/2009 12:10:26 PM

TIME	12:10	12:40	13:10	13:40	14:10	14:40	15:10	15:40	16:10	16:40	17:10	17:40	18:10	18:40	19:10	19:40	20:10
FRACT. DEF.	0.220	0.260	0.120	0.240	0.200	0.200	0.180	0.040	0.120	0.080	0.020	0.080	0.240	0.080	0.160	0.080	0.220
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Fraction Defective Parts Chart using time stamp labeling of the x-axis

Set the x-axis labeling mode using the overall charts setXAxisStringLabelMode method, setting it `SPCChartObjects.AXIS_LABEL_MODE_TIME.`

```
// Label the tick mark with time stamp of sample group
```

```
this.setXAxisStringLabelMode (SPCChartObjects.AXIS_LABEL_MODE_TIME);
```

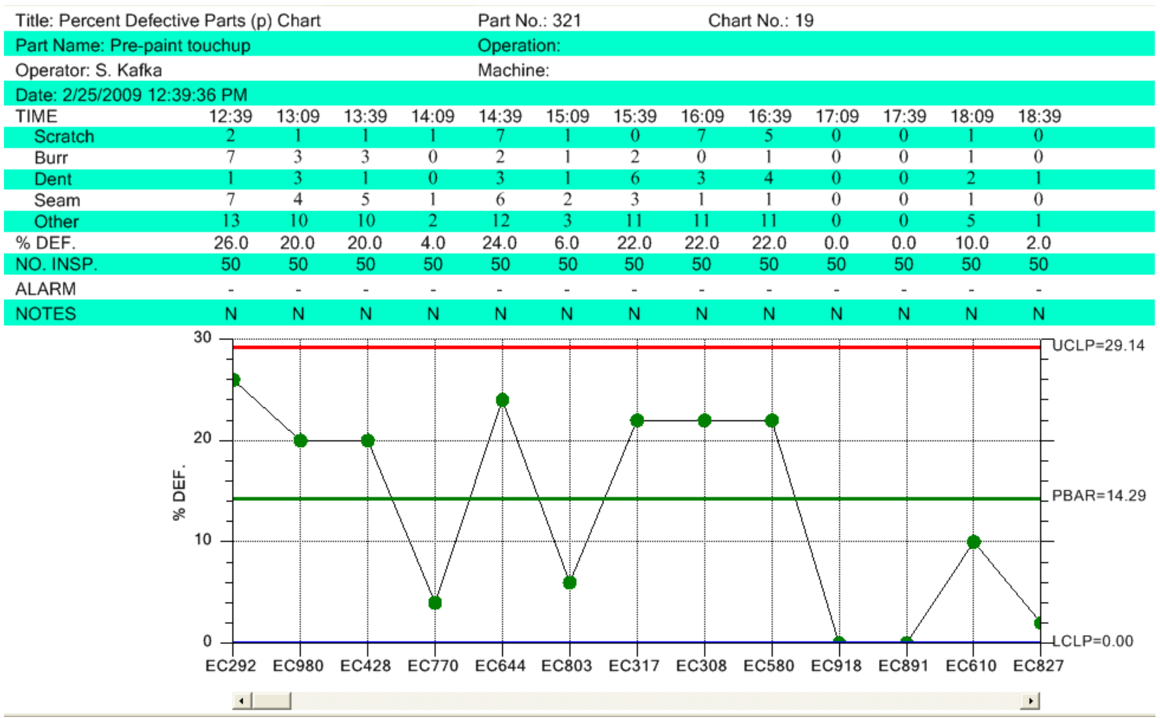
When updating the chart with sample data, use addNewSampleRecord overload that has batch number and a time stamp parameters.

```
this.getChartData().addNewSampleRecord(batchCounter, timestamp, samples);
```

See the example program BatchAttributeControlCharts.FractionDefectivePartsControlChart for a complete example. Reset the axis labeling mode back to batch number labeling by calling the setXAxisStringLabelMode method with SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

```
this.setXAxisStringLabelMode( SPCChartObjects.AXIS_LABEL_MODE_DEFAULT);
```

Batch Control Chart X-Axis User-Defined String Labeling



Title: Percent Defective Parts (p) Chart   Part No.: 321   Chart No.: 19
Part Name: Pre-paint touchup   Operation:
Operator: S. Kafka   Machine:
Date: 2/25/2009 12:39:36 PM

TIME	12:39	13:09	13:39	14:09	14:39	15:09	15:39	16:09	16:39	17:09	17:39	18:09	18:39
Scratch	2	1	1	1	7	1	0	7	5	0	0	1	0
Burr	7	3	3	0	2	1	2	0	1	0	0	1	0
Dent	1	3	1	0	3	1	6	3	4	0	0	2	1
Seam	7	4	5	1	6	2	3	1	1	0	0	1	0
Other	13	10	10	2	12	3	11	11	11	0	0	5	1
% DEF.	26.0	20.0	20.0	4.0	24.0	6.0	22.0	22.0	22.0	0.0	0.0	10.0	2.0
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N

Percent Defective Parts Chart using user-defined string labeling of the x-axis

Set the x-axis labeling mode using the overall charts setXAxisStringLabelMode method, setting it `SPCChartObjects.AXIS_LABEL_MODE_STRING.`

```
// enable scroll bar
this.setEnableScrollBar( true);
this.setEnableCategoryValues(false);
```

```
// Label the tick mark with user-defined strings
this.setXAxisStringLabelMode (SPCChartObjects.AXIS_LABEL_MODE_STRING);
```

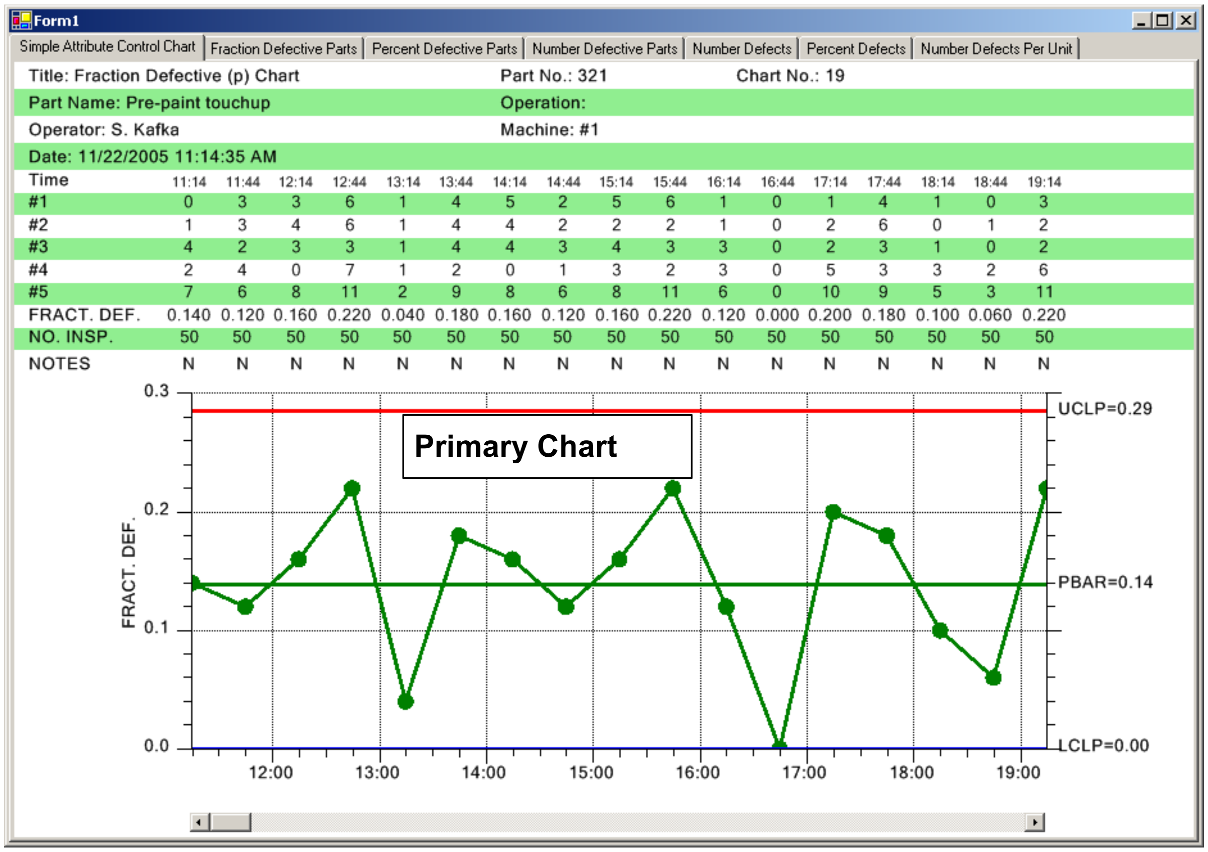
Use the addAxisUserDefinedString method to supply a new string for every new sample subgroup. It must be called every time the addNewSampleRecord method is called, or the user-defined strings will get out of sync with their respective sample subgroup. . Reset the axis labeling mode back to batch number labeling by calling the setXAxisStringLabelMode method with SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

```
this.getChartData().addNewSampleRecord(batchCounter, timestamp, samples,
variableControlLimits);

// Make a random string to simulate some sort of batch sample group ID
int randomnum= (int) (1000 * ChartSupport.getRandomDouble());
String batchidstring = "EC" + randomnum.ToString();
this.getChartData().addAxisUserDefinedString(batchidstring);
```

See the example program BatchAttributeControlCharts.PercentDefectivePartsControlChart for a complete example.

Changing Default Characteristics of the Chart

All *Attribute Control Charts* have one distinct graph with its own set of properties. This graph is the Primary Chart.



Title: Fraction Defective (p) Chart   Part No.: 321   Chart No.: 19
Part Name: Pre-paint touchup   Operation:
Operator: S. Kafka   Machine: #1
Date: 11/22/2005 11:14:35 AM

Time	11:14	11:44	12:14	12:44	13:14	13:44	14:14	14:44	15:14	15:44	16:14	16:44	17:14	17:44	18:14	18:44	19:14
#1	0	3	3	6	1	4	5	2	5	6	1	0	1	4	1	0	3
#2	1	3	4	6	1	4	4	2	2	2	1	0	2	6	0	1	2
#3	4	2	3	3	1	4	4	3	4	3	3	0	2	3	1	0	2
#4	2	4	0	7	1	2	0	1	3	2	3	0	5	3	3	2	6
#5	7	6	8	11	2	9	8	6	8	11	6	0	10	9	5	3	11
FRACT. DEF.	0.140	0.120	0.160	0.220	0.040	0.180	0.160	0.120	0.160	0.220	0.120	0.000	0.200	0.180	0.100	0.060	0.220
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Logically enough, the properties of the objects that make up each of these graphs are stored in a property named PrimaryChart. Once the graph is initialized (using the **initSPCTimeAttributeControlChart**, or **initSPCBatchAttributeControlChart** method), you can modify the default characteristics of each graph using these properties.

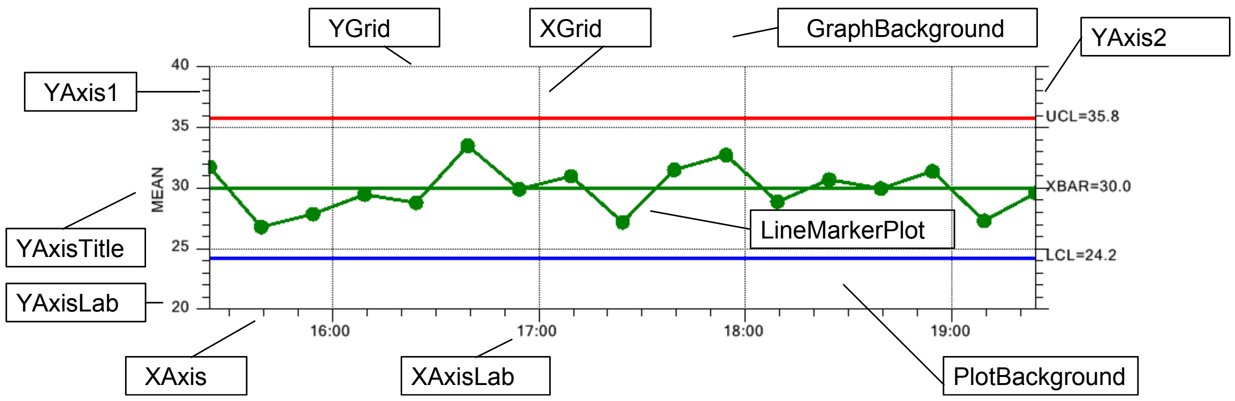
```
// Initialize the SPCTimeAttributeControlChart
this.initSPCTimeAttributeControlChart(charttype, numcategories,
    numsamplespersubgroup, numdatapointsinview, timeincrementminutes);
.
.
.

this.setGraphStartPosX ( 0.2);
// Enable scroll bar
this.setEnableScrollBar ( true);

SPCChartObjects primarychart = this.getPrimaryChart();

primarychart.setDisplayFrequencyHistogram ( true);

primarychart.getXAxis().setLineColor ( ChartColors.BLUE);
primarychart.getXAxis().setLineWidth ( 1);


primarychart.getProcessVariableData().getLineMarkerPlot().setLineColor( ChartColor
s.BLACK);

primarychart.getProcessVariableData().getLineMarkerPlot().getSymbolAttributes().se
tPrimaryColor ( ChartColors.BLUEVIOLET);

primarychart.getProcessVariableData().getLineMarkerPlot().getSymbolAttributes().se
tFillColor (ChartColors.BEIGE);


primarychart.getGraphBackground().setFillColor ( ChartColors.LIGHTGREY);

primarychart.getPlotBackground().setFillColor ( ChartColors.LIGHTGOLDENRODYELLOW);

primarychart.getFrequencyHistogramChart().getPlotBackground().setFillColor (
ChartColors.LIGHTGOLDENRODYELLOW);
```

The **PrimaryChart** object is an instance of the **SPCChartObjects** class. The **SPCChartObjects** class contains the objects needed to display a single graph. Below you will find a summary of the class properties.

Class SPCChartObjects – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/SPCChartBase.html

a link to the JavaDoc documentation for the class **SPCChartObjects**

The main objects of the graph are labeled in the graph below.



8. Named and Custom Control Rule Sets

Western Electric (WECO) Rules

The normal SPC control limit rules display at the 3-sigma level, both high and low. In this case, a simple threshold test determines if a process is in, or out of control. Once a process is brought under control using the simple 3-sigma level tests, quality engineers often want to increase the sensitivity of the control chart, detecting and correcting problems before the 3-sigma control limits are reached. Other, more complex tests rely on more complicated decision-making criteria. These rules utilize historical data and look for a non-random pattern that can signify that the process is out of control, before reaching the normal +-3 sigma limits. The most popular of these are the Western Electric Rules, also know as the WECO Rules, or WE Runtime Rules. First implemented by the Western Electric Co. in the 1920's, these quality control guidelines were codified in the 1950's and form the basis for all of the other rule sets. Different industries across the globe have have developed their own variants on the WECO Rules. Other sets of rules, common enough to have an identifying name, i.e. *named rules*, are listed below.

WECO Runtime and Supplemental Rules – Western Electric Co. - Western Electric Company (1956), *Statistical Quality Control handbook.* (1 ed.), Indianapolis, Indiana: Western Electric Co., p. v, OCLC 33858387. Sometimes the Supplemental Rules are referred to as the Montgomery Rules, after the statistical quality control expert Douglas Mongtomery. *Introduction to Statistical Quality Control* (5 ed.), Hoboken, New Jersey: John Wiley & Sons, ISBN 9780471656319

Nelson Rules – The Nelson rules were first published in the October 1984 issue of the Journal of Quality Technology in an article by Lloyd S Nelson.

AIAG Rules– The (AIAG) Automotive Industry Action Group control rules are published in the their industry group “Statistical Process Control Handbook”.

Juran Rules - Joseph M. Juran was an international expert in quality control and defined these rules in his “Juran's Quality Handbook”, McGraw-Hill Professional; 6 edition (May 19, 2010), **ISBN-10:** 0071629734

Hughes Rules – The only sources we could find for the Hughes rules were all second hand. If anyone can direct is to an original source for the Hughes Rules, please send an e-mail to support@quinn-curtis.com.

Duncan Rules – Acheson Johnston Duncan was an international expert in quality control and published his rules in the text book “Quality control and industrial statistics” (fifth edition). Irwin, 1986.

Gitlow Rules - Dr. Howard S. Gitlow is an international expert in Sigma Six, TQM and SPC. His rules are found in his book “Tools and Methods for the Improvement of Quality”, 1989, **ISBN-10: 0256056803** .

Westgard Rules – The Westgard rules are based on the work of James Westgard, a leading expert in laboratory quality management . They are considered “Laboratory quality control rules”. You can find more information about the Westgard Rules, and James Westgard at the web site: http://www.westgard.com

The rules sets have many individual rules in common. In particular, the WECO rules and the Nelson rules, have 7 out of 8 rules in common, and only differ in the fourth rule.

Western Electric (WECO) Rules

In the Western Electric Rules A process is considered out of control if any of the following criteria are met:

1. **The most recent point plots outside one of the 3-sigma control limits.** If a point lies outside either of these limits, there is only a 0.3% chance that this was caused by the normal process.

2. **Two of the three most recent points plot outside and on the same side as one of the 2-sigma control limits.** The probability that any point will fall outside the warning limit is only 5%. The chances that two out of three points in a row fall outside the warning limit is only about 1%.

3. **Four of the five most recent points plot outside and on the same side as one of the 1-sigma control limits.** In normal processing, 68% of points fall within one sigma of the mean, and 32% fall outside it. The probability that 4 of 5 points fall outside of one sigma is only about 3%.

4. **Eight out of the last eight points plot on the same side of the center line, or target value.** Sometimes you see this as 9 out of 9, or 7 out of 7. There is an equal chance that any given point will fall above or below the mean. The chances that a point falls on the same side of the mean as the one before it is one in two. The odds that the next point will also fall on the same side of the mean is one in four. The probability of getting eight points on the same side of the mean is only around 1%.

These rules apply to both sides of the center line at a time. Therefore, there are eight actual alarm conditions: four for the above center line sigma levels and four for the below center line sigma levels.

There are also additional WE Rules for trending. These are often referred to as WE Supplemental Rules. Don't rely on the rule number, often these are listed in a different order.

5. **Six points in a row increasing or decreasing.** The same logic is used here as for rule 4 above. Sometimes this rule is changed to seven points rising or falling.

6. **Fifteen points in a row within one sigma.** In normal operation, 68% of points will fall within one sigma of the mean. The probability that 15 points in a row will do so, is less than 1%.

7. **Fourteen points in a row alternating direction.** The chances that the second point is always higher than (or always lower than) the preceding point, for all seven pairs is only about 1%.

8. **Eight points in a row outside one sigma.** Since 68% of points lie within one sigma of the mean, the probability that eight points in a row fall outside of the one-sigma line is less than 1%.

The rules are described as they appear in the literature. In many cases, a given rule actually specifies two test conditions; the first being a value N out of M above a plus sigma control limit, and the second being a value N out of M below a minus sigma control limit. Examples of this are rules #1, #2 and #3 for WECO and Nelson rules. In other cases, similar rules only contain one test case; N out of M above (or below) a given sigma control limit. Example of this are the Juran rules #2..#5, Hughes Rules #2..#9, Gitlow Rules #2..#5, and Duncan Rules #2..#5.

While the list of named rules below follow what is presented in the literature, the actual rule numbering should be ignored. That is because in the software we implement all rules as simple single condition rules. The first rule in all of the named rule sets is implemented as two rules; a single point greater than 3-sigma; and a single point less than -3-sigma. And WECO and Nelson rules #2 and #3 are implemented as four rules; two N out of M greater than x-sigma condition limits, and two N out of M less than x-sigma condition limits. A complete cross reference to the named rules listed below, and our own rule number system is found in Table 1. This is important, because when you try to access a particular named rule within the software, you must use our rule number system.

Nelson Rules

The Nelson rules are almost identical to the combination of the WECO Runtime and Supplemental Rules. The only difference is in Rule #4.

4. Nine out of the last nine points plot on the same side of the center line, or target value.

AIAG Rules

1. One of one point is outside of 3-sigma control limit
2. Seven out of seven are above or below center line
3. Seven points in a row increasing
4. Seven points in a row decreasing

Juran Rules

1. One of one point is outside of +- 3-sigma control limit
2. Two of three points above 2-sigma control limit
3. Two of three points below -2-sigma control limit
4. Four of five points is above 1-sigma control limit
5. Four of five points is below -1-sigma control limit

6. Six points in a row increasing
7. Six points in a row decreasing
8. Nine out of nine are above or below center line
9. Eight points in a row on both sides of center line, none in zone C

Hughes Rules

1. One of one point is outside of +- 3-sigma control limit
2. Two of three points above 2-sigma control limit
3. Two of three points below -2-sigma control limit
4. Three of seven points above 2-sigma control limit
5. Three of seven points below -2-sigma control limit
6. Four of ten points above 2-sigma control limit
7. Four of ten points below -2-sigma control limit
8. Four of five points is above 1-sigma control limit
9. Four of five points is below -1-sigma control limit
10. Seven points in a row increasing
11. Seven points in a row decreasing
12. Ten of eleven are above center line
13. Ten of eleven are below center line
14. Twelve of fourteen are above center line
15. Twelve of fourteen are below center line

Gitlow Rules

1. One of one point is outside of +- 3-sigma control limit
2. Two of three points above 2-sigma control limit
3. Two of three points below -2-sigma control limit
4. Four of five points is above 1-sigma control limit
5. Four of five points is below -1-sigma control limit
6. Eight points in a row increasing
7. Eight points in a row decreasing
8. Eight out of Eight are above center line
9. Eight out of Eight are below center line

Duncan Rules

1. One of one point is outside of +- 3-sigma control limit

2. Two of three points above 2-sigma control limit
3. Two of three points below -2-sigma control limit
4. Four of five points is above 1-sigma control limit
5. Four of five points is below -1-sigma control limit
6. Seven points in a row increasing
7. Seven points in a row decreasing

Westgard Rules

1. One of one point is outside of +- 3-sigma control limits - 13s
2. Two of two points outside +-2-sigma control limits - 22s
3. Four of four points outside +-1-sigma control limits - 41s
4. Ten of ten points on one side of center line - 10x
5. Two adjacent points on opposite sides of +-2-sigma - R4s
6. Seven of seven points in a trend increasing or decreasing - 7T
7. One of one point is outside of +- 2-sigma control limits – 12s
8. Two of three points outside +-2-sigma control limits - 2of32s
9. Three of three points outside +-1-sigma control limits - 31s
10. Six of six points on one side of center line - 6x
11. Eight of eight points on one side of center line - 8x
12. Nine of nine points on one side of center line - 9x
13. Twelve of twelve points on one side of center line – 12x

By default, only the first six Westgard rules described above are enabled. The others can be turned on using the **UseNamedRuleSet** method and setting *ruleflags* array elements true for the additional rules. Make sure you use our rule numbers and not the rule numbering above.

Control Rule Templates

All of the named rules fall into one of our standard rule categories. Each rule category is a flexible template which can be used to evaluate a test condition across a wide range of parameters. A list of the template categories appears below.

Standardized Templates for Control Rule Evaluation

Template #	
	Standard Control Limit tests
1	N of M above X sigma (from center line), used for UCL tests
2	N of M below X sigma (from center line), used for LCL tests
3	Reserved
4	N of M beyond X sigma (from center line, either side) or control limits – points beyond the +- limit values – don't have to all be on one side

	Trending
5	N of M trending up (increasing)
6	N of M trending down (decreasing)
7	N of M trending up (increasing) or down (decreasing)
	Hugging (lack of variance)
8	N of M within X sigma (from center line, either side)
9	N of M within X sigma of each other (no reference to center line)
	Oscillation
10	N of M alternating about X sigma (from center line)
11	N of M alternating (no reference to center line)

For example, rule #1 for all of the named rules (a single point plots outside of +- 3 sigma) is implemented as one instance of template #1 (N of M above X sigma, where N=1, M=1 and X = 3) and one instance of template #2 (N of M below X sigma) where N=1, M=1 and X = -3).

Rule #2 for WECO and Nelson (two of three point plots outside of +- 2 sigma) is implemented as one instance of template #1 (N of M above X sigma, where N=2, M=3 and X = 2) and one instance of template #2 (N of M below X sigma) where N=2, M=3 and X = -2).

Rule #4 and #5 for Hughes (three of seven points above/below 2-sigma control limit) is implemented as one instance of template #1 (N of M above X sigma, where N=3, M=7 and X = 2) and one instance of template #2 (N of M below X sigma) where N=3, M=7 and X = -2).

Rule #6 for Gitlow (eight points in a row increasing) is implemented as one instance of template #5 (N of M trending up) where N=8 and M=8.

The templates are important because using them you can modify any existing named rule, changing the M, N or X parameter. Or, you can create completely new rules.

Taking these factors into account, we have redefined and renumbered the rules, identifying each with the template and parameters used by each rule, .

Standardized Template Parameters and Rule # Cross Reference for Named Rules

Basic Rules		**New**				
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3

WECO **New**

Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
	2 of 3 > 2 sigma	4	2	2	3	2
#3	4 of 5 < sigma	5	1	4	5	-1
	4 of 5 > sigma	6	2	4	5	1
#4	8 of 8 < center line	7	1	8	8	0
	8 of 8 > center line	8	2	8	8	0

WECO+Supplemental		**New**				
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
	2 of 3 > 2 sigma	4	2	2	3	2
#3	4 of 5 < sigma	5	1	4	5	-1
	4 of 5 > sigma	6	2	4	5	1
#4	8 of 8 < center line	7	1	8	8	0
	8 of 8 > center line	8	2	8	8	0
#5	6 of 6 incr. or dec.	9	7	6	6	0
#6	15 of 15 within 1 sigma	10	8	15	15	1
#7	14 of 14 alternating	11	11	14	14	0
#8	8 of 8 outside zone C	12	4	8	8	1

Nelson		**New**				
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
	2 of 3 > 2 sigma	4	2	2	3	2
#3	4 of 5 < sigma	5	1	4	5	-1
	4 of 5 > sigma	6	2	4	5	1
#4	9 of 9 < center line	7	1	9	9	0
	9 of 9 > center line	8	2	9	9	0
#5	6 of 6 incr. or dec.	9	7	6	6	0
#6	15 of 15 within 1 sigma	10	8	15	15	1
#7	14 of 14 alternating	11	11	14	14	0
#8	8 points outside zone C	12	4	8	8	1

AIAG		**New**				
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**

#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	7 of 7 < center line	3	1	7	7	0
#3	7 of 7 > center line	4	2	7	7	0
#4	7 of 7 increasing	5	5	7	7	0
#5	7 of 7 decreasing	6	6	7	7	0

Juran Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	4 of 5 < sigma	5	1	4	5	-1
#5	4 of 5 > sigma	6	2	4	5	1
#6	6 of 6 increasing	7	5	6	6	0
#7	6 of 6 decreasing	8	6	6	6	0
#8	9 of 9 > center line	9	1	9	9	0
#9	9 of 9 < center line	10	2	9	9	0
#10	8 of 8 outside zone C	11	4	8	8	1

Hughes Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	3 of 7 < 2 sigma	5	1	3	7	-2
#5	3 of 7 > 2 sigma	6	2	3	7	2
#6	4 of 10 < 2 sigma	7	1	4	10	-2
#7	4 of 10 > 2 sigma	8	2	4	10	2
#8	4 of 5 < sigma	9	1	4	5	-1
#9	4 of 5 > sigma	10	2	4	5	1
#10	7 of 7 increasing	11	5	7	7	0
#11	7 of 7 decreasing	12	6	7	7	0
#12	10 of 11 < center line	13	1	10	11	0
#13	10 of 11 > center line	14	2	10	11	0
#14	12 of 14 < center line	15	1	12	14	0
#15	12 of 14 > center line	16	2	12	14	0

Gitlow Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3

#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	4 of 5 < sigma	5	1	4	5	-1
#5	4 of 5 > sigma	6	2	4	5	1
#6	8 of 8 increasing	7	5	8	8	0
#7	8 of 8 decreasing	8	6	8	8	0
#8	8 of 8 < center line	9	1	8	8	0
#9	8 of 8 > center line	10	2	8	8	0

Duncan Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	4 of 5 < sigma	5	1	4	5	-1
#5	4 of 5 > sigma	6	2	4	5	1
#6	7 of 7 increasing	7	5	7	7	0
#7	7 of 7 decreasing	8	6	7	7	0

Westgard Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
2	2 of 2 < 2 sigma	3	1	2	2	-2
	2 of 2 > 2 sigma	4	2	2	2	2
3	4 of 4 < 1 sigma	5	1	4	4	-1
	4 of 4 > 1 sigma	6	2	4	4	1
4	10 of 10 < centerline	7	1	10	10	0
	10 of 10 > centerline	8	2	10	10	0
5	R2s – 2-sigma limits	9	10	1	1	2
6	7 of 7 trending	10	7	7	7	0
7	1 of 1 > 2 sigma	11	1	1	1	-2
	1 of 1 < 2 sigma	12	2	1	1	2
8	2 of 3 > 3 sigma	13	1	2	3	-2
	2 of 3 < 3 sigma	14	2	2	3	2
9	3 of 3 > 1 sigma	15	1	3	3	-1
	3 of 3 < 1 sigma	16	2	3	3	1
10	6 of 6 < centerline	17	1	6	6	0
	6 of 6 > centerline	18	2	6	6	0
11	8 of 8 < centerline	19	1	8	8	0
	8 of 8 > centerline	20	2	8	8	0
12	9 of 9 < centerline	21	1	9	9	0

	9 of 9 > centerline	22	2	9	9	0
13	12 of 12 < centerline	23	1	12	12	0
	12 of 12 > centerline	24	2	12	12	0

Implementing a Named Rule Set

You are able to add a named rule set to an SPC application using a single call. Call the **useNamedRuleSet** method, passing in the appropriate rule ID.

SPCChartObjects.UseNamedRuleSet Method

```
public void useNamedRuleSet(
        int ruleset
)
```

```
public void useNamedRuleSet(
        int ruleset,
        BoolArray ruleflags
)
```

Parameters

ruleset

One of the SPCControlLimitRecord named rule indentifiers: BASIC_RULES, WECO_RULES,WECOANDSUPP_RULES, NELSON_RULES,AIAG_RULES, JURAN_RULES, HUGHES_RULES,GITLOW_RULES, WESTGARD_RULES, and DUNCAN_RULES.

ruleflags

An array of boolean, one for each named rule in the rule set. ///

```
// Use the standard +-3-sigma control limits. Equivalent to WECO control rules #1 and #2.
this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.BASIC_RULES);
```

```
// Use the WECO rules.  This corresponds WECO rules #1... #4, which in our organization of the
rules is #1 to #8

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.WECO_RULES);


// Use the WECO rules.  This corresponds WECO rules #1... #8 , which in our organization of the
rules is #1 to #12

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.WECOANDSUPP_RULES);


// Use the complete set of Nelson rules.

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.NELSON_RULES);


// Use the complete set of AIAG rules.

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.AIAG_RULES);


// Use the complete set of Juran rules.

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.JURAN_RULES);


// Use the complete set of Hughes rules.

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.HUGHES_RULES);



// Use the complete set of Gitlow rules.

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.GITLOW_RULES);


// Use the complete set of Duncan rules.

this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.DUNCAN_RULES);
```

The named rule setup methods also come in variants which allow you to selectively enable/diasble one or more rules from the named rule set. You do that by building a BoolArray object, containing true or false for each named rule in the rule set, and calling **useNamedRuleSet** method. The **getInitializedRuleBoolArray** is just a simple utility method which creates and fills out an appropriately sized BoolArray array with a default true, or false value.

SPCChartObjects.getInitializedRuleBoolArray Method

```
public BoolArray getInitializedRuleBoolArray(
        int ruleset,
        bool initialvalue
)
```

Parameters

ruleset

One of the SPCControlLimitRecord named rule identifiers: BASIC_RULES, WECO_RULES,WECOANDSUPP_RULES, NELSON_RULES,AIAG_RULES, JURAN_RULES, HUGHES_RULES,GITLOW_RULES, WESTGARD_RULES, and DUNCAN_RULES.

initialvalue

True or False, all values of the returned BoolArray are initialized to this value.

*All rule numbering is based on our rule numbering, which separates greater than and less than tests into separate rules, as detailed in the previous tables.

```
BoolArray ruleflags = this.getPrimaryChart().getInitializedRuleBoolArray(
   SPCControlLimitRecord.NELSON_RULES, true);
ruleflags[9] = false;
ruleflags[10] = false;
this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.NELSON_RULES , ruleflags);
```

The example above disables rule #9 (6 of 6 increasing or decreasing) and rule #10 (15 of 15 within 1 sigma) from the Nelson Rules. You use our rule numbering system for specifying which rule.

See the example program RulesRulesRules for examples of how to setup an SPC chart for a given set of control rules. Once you add a set of named control rules to your SPC chart, the next thing you will want to do is set the control limits. You can either set the limits using known values, or you can have our software calculate the limits using previously acquired sample data.

If you want to explicitly set the limits you must know the historical process mean (also called the center line) and the historical process sigma. You may already know your process sigma, or you may need to calculate it as 1/3 * (UCL – process mean), where UCL is your historical +3-sigma upper control limit. Once you have those two values, everything else is automatic. Just call the

updateControlLimitUsingMeanAndSigma method. It will run through all of the control limit records and fill out the appropriate limit values and other critical parameters.

SPCControlChartData.updateControlLimitsUsingMeanAndSigma Method

```
public void updateControlLimitsUsingMeanAndSigma(
        int chartpos,
        double mean,
        double sigma
)
```

Parameters

chartpos

specifiy either SPC_PRIMARY_CONTROL_TARGET or SPC_SECONDARY_CONTROL_TARGET

mean

specify the process mean.

sigma

specify the process sigma.

```
double processMean = your process mean
double processSigma = your process sigma
this.getChartData().updateControlLimitsUsingMeanAndSigma(SPCChartObjects.PRIMARY_CHART,
    processMean, processSigma);
```

The center line value and sigma have different meanings for the Primary and Secondary charts. So the **updateControlLimitsUsingMean** and Sigma applies to only one at a time. If you use it for the secondary chart control limits, use your historical center line value for the secondary chart type you are using. Calculate the sigma value as 1/3 * (UCL – center line), where UCL is your historical +3-sigma upper control limit for your secondary chart.

You can also auto-calculate the control limits by adding test data to your application (fed into the chart using the **addNewSampleRecord** method), and calling **autoCalculatedControlLimits.** This fills out the **SPCControlLimit** record for each control rule of the named rule set and makes control limit checking possible. You will find the AutoCalculateControlLimits method used in all of SPC charts of the RulesRulesRules example program.

```
this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.WECO_RULES);

// Must have data loaded before any of the Auto.. methods are called
 simulateData(100, 20);

// Calculate the SPC control limits for both graphs of the current SPC chart
this.autoCalculateControlLimits();
```

Modifying Existing Named Rules

Perhaps you like everything about a named rule set, except for one or more rules. For example, you want to use the Hughes rules, but want to change the N of M parameters of rules #15 and #16. You do that using the **SPCControlLimitRecord** method **setCustomRuleParameters**.

SPCControlLimitRecord.setCustomRuleParameters Method

```
public void setCustomRuleParameters(
        int valuesincalc,
        int numvaluesforruleviolation
)
```

Parameters

valuesincalc

Specifies the number of values to use in calculation.

numvaluesforruleviolation

Specifies the number of values that must be outside alarm limit for rule violation.

The example below changes the N of M parameters of Hughes rules #15 and #16 from their default N of M value (12 of 14), to the values (15 of 18).

```
this.getPrimaryChart().useNamedRuleSet(SPCControlLimitRecord.HUGHES_RULES);

int M = 18;
int N = 15;
SPCControlLimitRecord clr =
    this.getChartData().getNamedControlRuleRecord(SPCControlLimitRecord.HUGHES_RULES, 15);
clr.setCustomRuleParameters(M, N);
clr = this.getChartData().getNamedControlRuleRecord(SPCControlLimitRecord.HUGHES_RULES, 16);
clr.setCustomRuleParameters(M, N);
```

When trying to access the **SPCControlLimitRecord** for a given control rule, it is very important that you use the **getNamedControlRuleRecord** method, not the **getControlLimitRecord** method. This is because the array indexing of the **ControlLimitRecords** does not match the named control rule numbering, i.e., **getNamedControlRuleRecord**(15) and **getControlLimitRecord**(15) return different SPCControlLimitRecords.

If you want to enable/disable a specific rule, after they have all been created,

```
SPCControlLimitRecord clr =
       this.getChartData().getNamedControlRuleRecord(SPCControlLimitRecord.AIAG_RULES, 3);
clr.AlarmEnable = false;
```

Creating Custom Rules Sets Based on Named Rules

You can create your own custom set of rules, mixing and matching rules from the standard, named rule sets, using the the **AddControlRule** method. Also, you can invent your own rules, based on one of our standard templates, and add those rules to your custom rule set.

SPCChartObjects.addControlRule Method

```
public int addControlRule(
        int ruleset,
        int rulenum
)
```

Parameters

ruleset

One of the SPCControlLimitRecord named rule indentifiers: BASIC_RULES, WECO_RULES,WECOANDSUPP_RULES, NELSON_RULES,AIAG_RULES, JURAN_RULES, HUGHES_RULES,GITLOW_RULES, WESTGARD_RULES, and DUNCAN_RULES.

rulenum

The rule number (our rule number) ///

Even if you do not call one of the **use..Rules** methods, you still end up with four control limits. These correspond to the +-3-sigma control limits, for both the Primary and Secondary (were applicable) chart. So, you do not need to add those to your custom set of rules. Start with the rules you want to add after the standard +-3-sigma rules. If, for some reason you cannot live with the default +-3-sigma rules, you can disable them with a call to **enableDefaultLimits**(false, false).

Say you want to create custom rule set for the Primary chart, combining rules from Nelson (#3, #4), Juran (#5, #5), AIAG (#3,#4), Hughes (#12) and Duncan (#8). Make the following calls in the setup portion of your program.

```
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.NELSON_RULES, 3);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.NELSON_RULES, 4);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.JURAN_RULES, 5);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.JURAN_RULES, 6);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.AIAG_RULES, 3);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.AIAG_RULES, 4);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.HUGHES_RULES, 12);
this.getPrimaryChart().addControlRule(SPCControlLimitRecord.DUNCAN_RULES, 8);
```

Normally there will be no reason to set custom rules for the secondary chart, since all of the named rules apply to the Primary chart. Nothing stops you from doing it though. We can't say whether or not it makes sense statistically.

Creating Custom Rules Sets Based on a Template

Add your own custom rule to the rule set using another version of the **addControlRuleRecord** method. This one specifies a template, N out of M values, a sigma level to attach the control rule to, and a flag on whether or not to display a limit line for the control rule. If you have multiple control rules attached to a given sigma level, you should only display a control line for one of them.

SPCChartObjects.addControlRule Method

```
public int addControlRule(
        int template,
        int n,
        int m,
        double sigmavalue,
        bool displaylimitline
)
```

Parameters

template

Specifies the standardized template number.

n

The n value for the n of m condition ///

m

The m value for the n of m condition ///

sigmavalue

The sigma value associated with the limit ///

displaylimitline

True to display a limit line for the control limit ///

In your code it would something like:

```
int template = 2; // N of M  Greater than limit value
int N = 10;
int M = 13;
double sigmavalue = 2; // control limit value is the +2-sigma value
bool displaylimitline = false; // no limit line.
this.getPrimaryChart().addControlRule(template, N, M,
    sigmavalue, displaylimitline);
```

Creating Custom Rules Not Associated With Sigma Levels

Most of the preceding control rules are based on the mean and sigma of the current control chart. The trending rules (N of M increasing/ decreasing) are an exception, because they don't use the mean or sigma value anywhere in their evaluation. Regardless, since many of the named rules include trending rules, they are included with the previous section. There are a couple of other control rules not directly related to the mean and sigma value of the chart. The first is a simple numeric limit. The limit is meant to be independent of the mean and sigma level of the chart. It can also be considered a specification limit. We have three routines you can use to add a numeric control limit to a chart. The first, **addSpecLimit**, creates a specification limit which monitors the main variable of the chart, and compares its value to the specified numeric threshold. It will display in the chart as a line with a limit string to the right of the plot area.

SPCChartObjects.addSpecLimit Method

```
public SPCControlPlotObjectData addSpecLimit(
        int speclimittype,
        double value,
        string displaystring,
        ChartAttribute attrib
)
```

Parameters

speclimittype

Specifiy either SPCChartObjects.SPC_LOWER_SPEC_LIMIT or SPCChartObjects.SPC_UPPER_SPEC_LIMIT

value

Specifies the value of the specification limit.

displaystring

The optional display string displayed to the right of the spec limit line.

attrib

The line attributes of the spec limit line.

Return Value

The SPCControlPlotObjectData object of the specification limit.

```
this.getPrimaryChart().addSpecLimit(SPCChartObjects.SPC_LOWER_SPEC_LIMIT,

    18.3, "L SPEC", new ChartAttribute(ChartColors.GREEN, 3.0));
this.getPrimaryChart().addSpecLimit(SPCChartObjects.SPC_UPPER_SPEC_LIMIT,

    39.1, "H SPEC", new ChartAttribute(ChartColors.YELLOW, 3.0));
```

The second, **addNumericControlLimit**, creates a limit which monitors the value of one of the charts SPCCalculatedValueRecord object values. This is what you would use if you wanted to monitor a subgroup mean, range, sigma against some arbitrary control limit, since in the appropriate charts, the subgroup mean, range and sigma are SPCCalculatedValueRecord objects.

The type, and ordering of the SPCCalculatedValueRecord records is unique to each chart type. Below is a list of the chart types and the calculated values for each.

MEAN_RANGE_CHART
0. SPCCalculatedValueRecord.SPC_MEAN_CALC,
1. SPCCalculatedValueRecord.SPC_RANGE_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC

MEDIAN_RANGE_CHART
0. SPCCalculatedValueRecord.SPC_MEDIAN_CALC
1. SPCCalculatedValueRecord.SPC_RANGE_CALC;

MEAN_SIGMA_CHART
0. SPCCalculatedValueRecord.SPC_MEAN_CALC

1. SPCCalculatedValueRecord.SPC_STD_DEVIATION_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC

MEAN_SIGMA_CHART_VSS

0. SPCCalculatedValueRecord.SPC_MEAN_VSS_CALC
1. SPCCalculatedValueRecord.SPC_STD_DEVIATION_VSS_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC

MEAN_VARIANCE_CHART

0. SPCCalculatedValueRecord.SPC_MEAN_CALC
1. SPCCalculatedValueRecord.SPC_VARIANCE_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC

INDIVIDUAL_RANGE_CHART

0. SPCCalculatedValueRecord.SPC_INDIVIDUAL_COPY_VALUE
1. SPCCalculatedValueRecord.SPC_INDIVIDUAL_ABS_RANGE_CAL

MAMR_CHART

0. SPCCalculatedValueRecord.SPC_MA_CALC
1. SPCCalculatedValueRecord.SPC_MR_CALC

MAMS_CHART

0. SPCCalculatedValueRecord.SPC_MA_CALC
1. SPCCalculatedValueRecord.SPC_MS_CALC

EWMA_CHART

0. SPCCalculatedValueRecord.SPC_EWMA_CALC
1. SPCCalculatedValueRecord.SPC_MEAN_CALC

MA_CHART

0. SPCCalculatedValueRecord.SPC_MA_CALC

TABCUSUM_CHAR

0. SPCCalculatedValueRecord.SPC_CUSUM_CPLUS_CALC
1. SPCCalculatedValueRecord.SPC_CUSUM_CMINUS_CALC
2. SPCCalculatedValueRecord.SPC_MEAN_CALC, "MEAN"

PERCENT_DEFECTIVE_PARTS_CHART)

0. SPCCalculatedValueRecord.SPC_PERCENT_DEFECTIVE_PARTS_CALC

FRACTION_DEFECTIVE_PARTS_CHART

0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTIVE_PARTS_CALC

FRACTION_DEFECTIVE_PARTS_CHART_VSS

0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTIVE_PARTS_VSS_CALC

NUMBER_DEFECTIVE_PARTS_CHART

0. SPCCalculatedValueRecord.SPC_TOTAL_DEFECTIVE_PARTS_CALC

NUMBER_DEFECTS_CHART

0. SPCCalculatedValueRecord.SPC_TOTAL_DEFECTS_CALC

NUMBER_DEFECTS_PERUNIT_CHART

0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTS_CALC

NUMBER_DEFECTS_PERUNIT_CHART_VSS

0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTS_VSS_CALC

PERCENT_DEFECTIVE_PARTS_CHART_VSS

0. SPCCalculatedValueRecord.SPC_PERCENT_DEFECTIVE_PARTS_VSS_CALC

NUMBER_DEFECTS_PER_MILLION_CHART

0. SPCCalculatedValueRecord.SPC_TOTAL_DEFECTS_CALC
1. SPCCalculatedValueRecord.SPC_NUMBER_DEFECTS_PER_MILLION_CALC

SPCChartObjects.addNumericControlLimit Method

```
public SPCControlPlotObjectData addNumericControlLimit(
        SPCCalculatedValueRecord sourcevar,
        int limtype,
        double value,
        string displaystring,
        bool addline,
        ChartAttribute attrib
)
```

Parameters

sourcevar

The SPCCalculatedValue source of the item to test.

limtype

Specifiy either SPCChartObjects.SPC_LOWERTHAN_LIMIT or SPCChartObjects.SPC_GREATERTHAN_LIMIT

value

Specifies the value of the control limit.

displaystring

The optional display string displayed to the right of the limit line.

addline

True and the limit is displayed as a line in the chart.

attrib

The line attributes of the spec limit line.

Return Value

The SPCControlPlotObjectData object of the numeric limit.

```
// this.getChartData().getCalculatedValueRecord(0) returns the calcuated value which calculates
the subgroup mean

SPCCalculatedValueRecord cvr = this.getChartData().getCalculatedValueRecord(0);
this.getPrimaryChart().addNumericControlLimit(cvr, SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
       22.3, "L CV(0)", true, new ChartAttribute(ChartColors.GREEN, 3.0));
this.getPrimaryChart().addNumericControlLimit(cvr, SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
       32.1, "H  CV(0)", true, new ChartAttribute(ChartColors.YELLOW, 3.0));
```

The third, **addProcessCapabilityLimit**, creates a limit which monitors the value of one of the charts SPCProcessCapabilityRecord object values.

SPCChartObjects.addProcessCapabilityControlLimit Method

```
public SPCControlPlotObjectData AddProcessCapabilityControlLimit(
        int pcindex,
        int limtype,
        double value,
        string displaystring,
        ChartAttribute attrib
)
```

Parameters

pcindex

Specify the process capability limit index (based on the order the process capabilities were added to the chart, starting at 0.

limtype

Specifiy either SPCChartObjects.SPC_LOWER_PC_LIMIT or SPCChartObjects.SPC_UPPER_PC_LIMIT

value

Specifies the value of the limit.

displaystring

Text included with the alarm if it is triggered.

attrib

The line attributes of the spec limit line.

Return Value

The SPCControlPlotObjectData object of the specification limit.

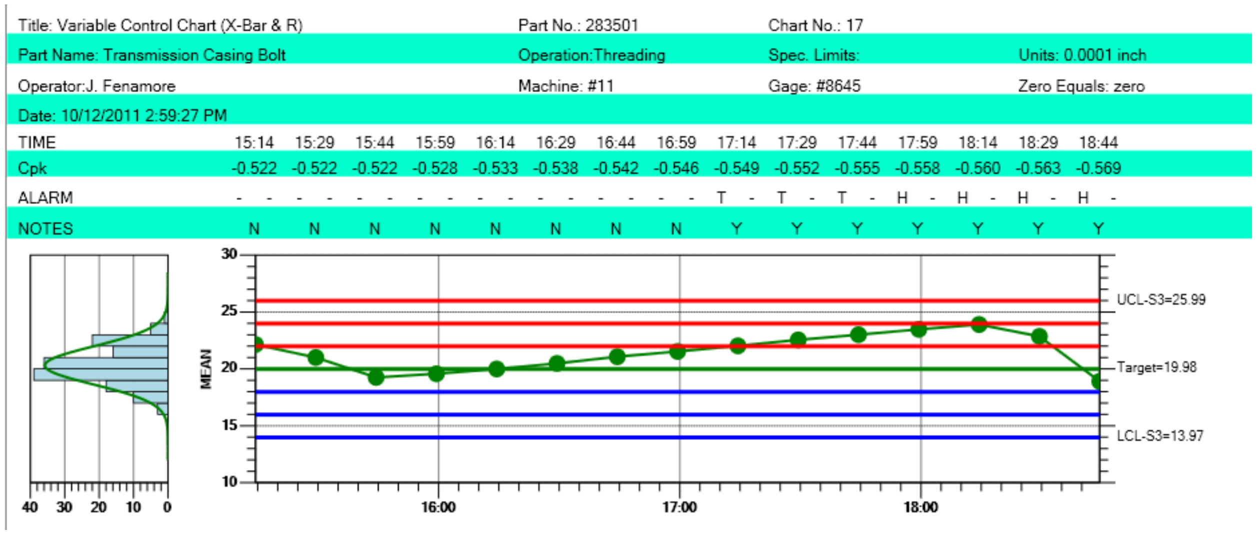
```
// Attached  control limits to the first Process Capability added to the chart

this.getPrimaryChart().addProcessCapabilityControlLimit(0, SPCChartObjects.SPC_LOWER_PC_LIMIT,
    0.195, "L Cpk", new ChartAttribute(ChartColors.GREEN, 3.0));

this.getPrimaryChart().addProcessCapabilityControlLimit(0, SPCChartObjects.SPC_UPPER_PC_LIMIT,
    0.25, "H Cpk", new ChartAttribute(ChartColors.YELLOW, 3.0));


// Attached  control limits to the second Process Capability added to the chart

this.getPrimaryChart().addProcessCapabilityControlLimit(1, SPCChartObjects.SPC_LOWER_PC_LIMIT,
    0.195, "L Cpm", new ChartAttribute(ChartColors.GREEN, 3.0));

this.getPrimaryChart().addProcessCapabilityControlLimit(1, SPCChartObjects.SPC_UPPER_PC_LIMIT,
    0.25, "H Cpm", new ChartAttribute(ChartColors.YELLOW, 3.0));
```

Enable Alarm Highlighting

The alarm status line above is turned on/off using the **EnableAlarmStatusValues** property. We have set it on by default, so you will have to turn it off if you don't want it. Each sample interval has two small boxes that are labeled using one of several different characters, listed below. The most common are an "H" signifying a high alarm, a "L" signifying a low alarm, and a "-" signifying that there is no alarm. When specialized control rules are implemented, using the named rules, or custom rules involving trending, oscillation, or stratification, a "T", "O" or "S" may also appear.

"-"	No alarm condition
"H"	High - Measured value is above a high limit
"L"	Low - Measured value falls below a low limit
"T"	Trending - Measured value is trending up (or down).
"O"	Oscillation - Measured value is oscillating (alternating) up and down.
"S"	Stratification - Measured value is stuck in a narrow band.

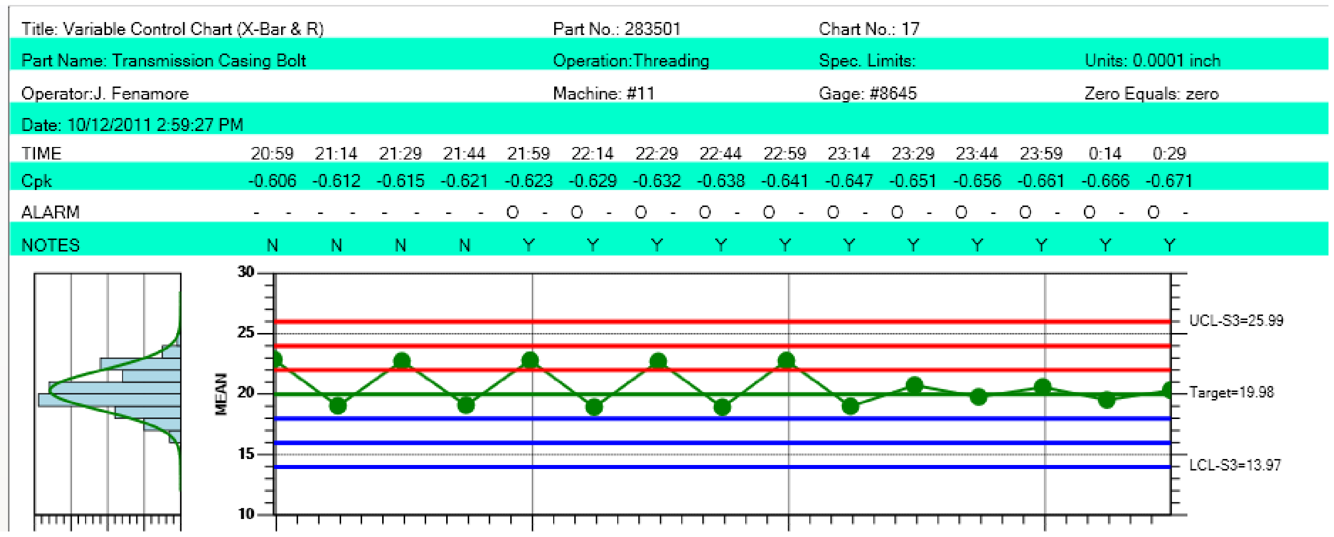
Trending



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation:Threading	Spec. Limits:	Units: 0.0001 inch
Operator:J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 10/12/2011 2:59:27 PM			

TIME	15:14	15:29	15:44	15:59	16:14	16:29	16:44	16:59	17:14	17:29	17:44	17:59	18:14	18:29	18:44
Cpk	-0.522	-0.522	-0.522	-0.528	-0.533	-0.538	-0.542	-0.546	-0.549	-0.552	-0.555	-0.558	-0.560	-0.563	-0.569
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	T -	T -	T -	H -	H -	H -	H -
NOTES	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y

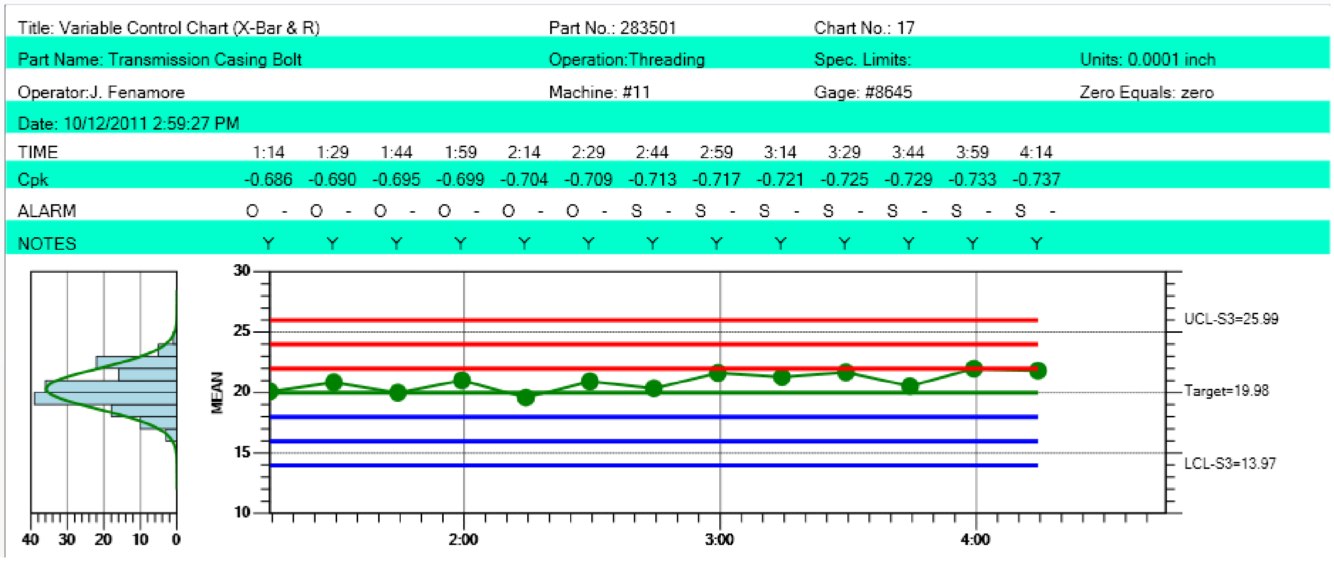
Oscillation



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation:Threading	Spec. Limits:	Units: 0.0001 inch
Operator:J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 10/12/2011 2:59:27 PM			

TIME	20:59	21:14	21:29	21:44	21:59	22:14	22:29	22:44	22:59	23:14	23:29	23:44	23:59	0:14	0:29
Cpk	-0.606	-0.612	-0.615	-0.621	-0.623	-0.629	-0.632	-0.638	-0.641	-0.647	-0.651	-0.656	-0.661	-0.666	-0.671
ALARM	- -	- -	- -	- -	O -	O -	O -	O -	O -	O -	O -	O -	O -	O -	O -
NOTES	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Stratification



Title: Variable Control Chart (X-Bar & R)	Part No.: 283501	Chart No.: 17	
Part Name: Transmission Casing Bolt	Operation:Threading	Spec. Limits:	Units: 0.0001 inch
Operator:J. Fenamore	Machine: #11	Gage: #8645	Zero Equals: zero
Date: 10/12/2011 2:59:27 PM			

TIME	1:14	1:29	1:44	1:59	2:14	2:29	2:44	2:59	3:14	3:29	3:44	3:59	4:14
Cpk	-0.686	-0.690	-0.695	-0.699	-0.704	-0.709	-0.713	-0.717	-0.721	-0.725	-0.729	-0.733	-0.737
ALARM	O -	O -	O -	O -	O -	O -	S -	S -	S -	S -	S -	S -	S -
NOTES	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

```
// Alarm status line
this.setEnableAlarmStatusValues(false);
```

Control Limit Alarm Event Handling

The **SPCControlChartData** class can throw an alarm event based on either the current alarm state, or an alarm transition from one alarm state to another. The **SPCControlLimitAlarmArgs** passes alarm data to the event handler. See *Chapter 5 - SPC Control Data and Alarm Classes,* for examples of using an alarm event handler.

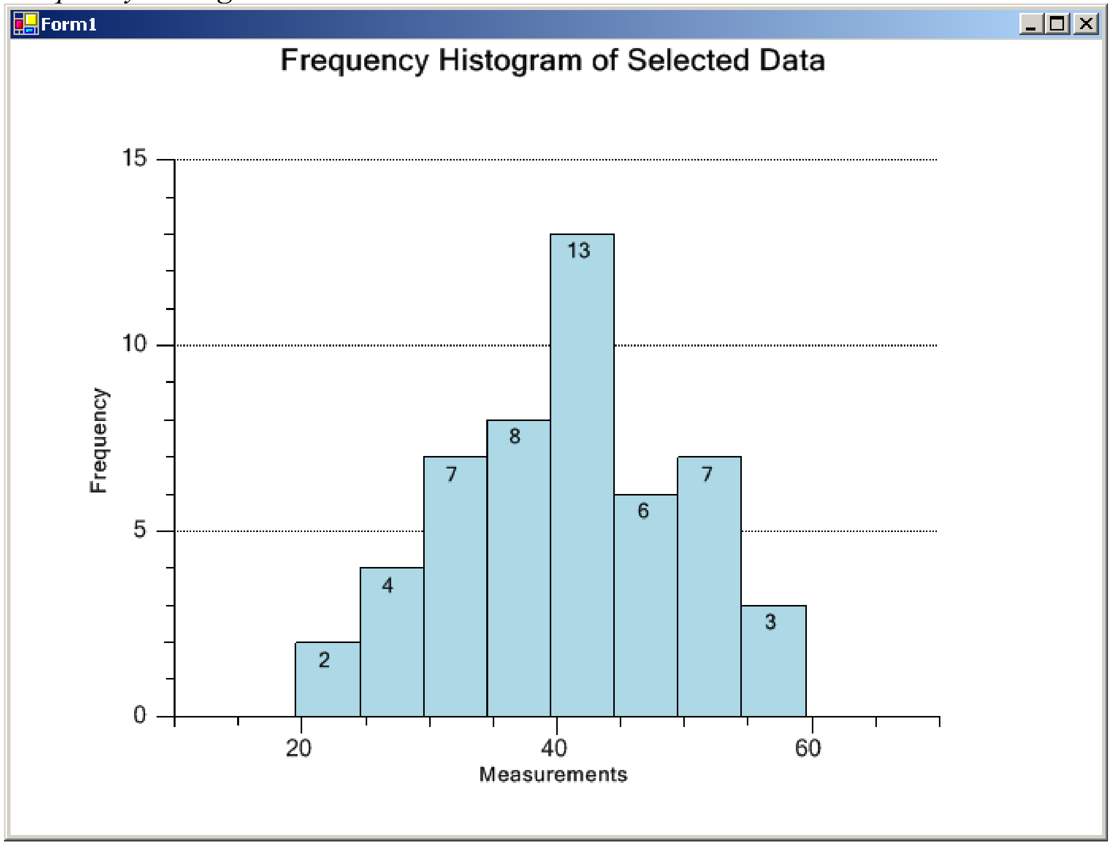
9. Frequency Histogram, Pareto Diagram and Normal-Probability Charts.

FrequencyHistogramChart
ParetoChart
ProbabilityChart

Frequency Histogram Chart

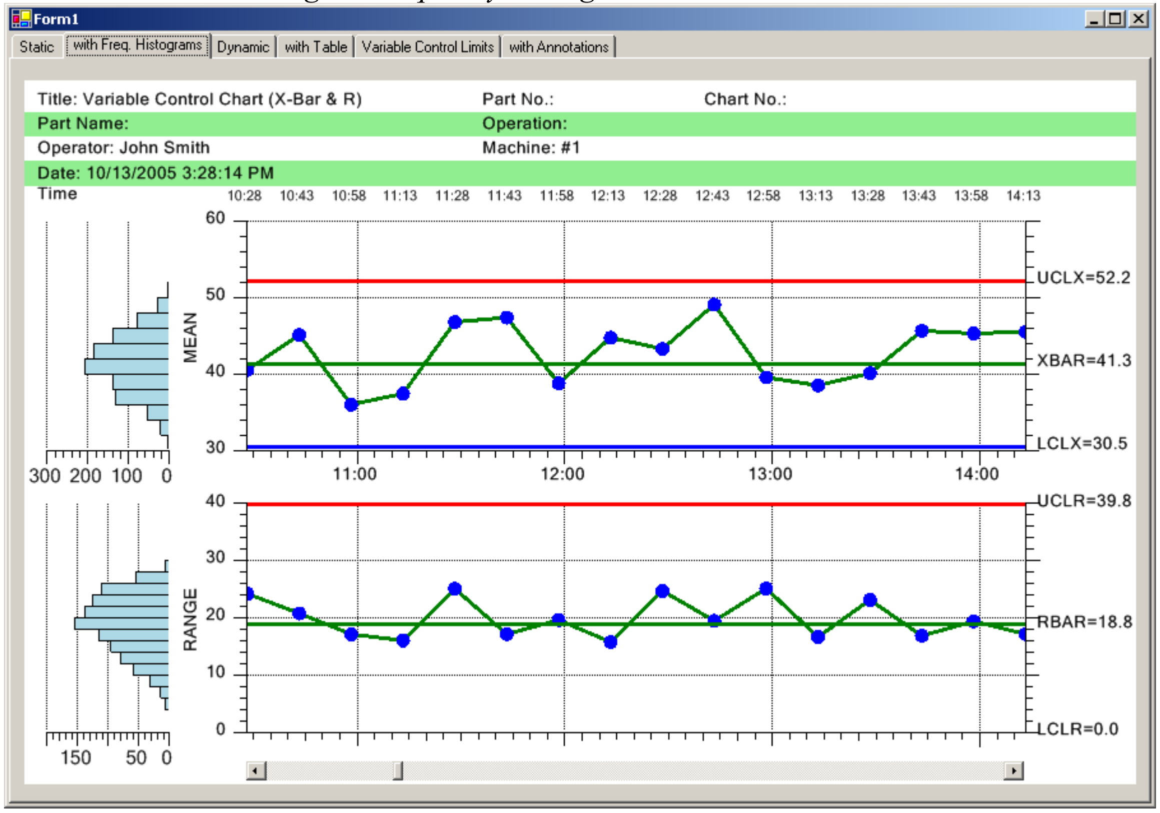
An SPC control chart will allow you to track the trend of critical variables in a production environment. It is important that the production engineer understand whether or not changes or variation in the critical variables are natural variations due to the tolerances inherent to the production machinery, or whether or not the variations are due to some systemic, assignable cause that needs to be addressed. If the changes in critical variables are due to the natural variations, then a frequency histogram of the variations will usually follow one of the common continuous (normal, exponential, gamma, Weibull) or discrete (binomial, Poisson, hypergeometric) distributions. It is the job of the SPC engineer to know what distribution best models his process. Periodically plotting of the variation of critical variables will give SPC engineer important information about the current state of the process. A typical frequency histogram looks like:

Frequency Histogram Chart



Viewing frequency histograms of both the variation in the primary variable (Mean, Median, count, or actual value), and the secondary variable (Range, Sigma or Moving Range) side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process.

XBar-R Chart with Integral Frequency Histograms



Title: Variable Control Chart (X-Bar & R) Part No.: Chart No.:
Part Name: Operation:
Operator: John Smith Machine: #1
Date: 10/13/2005 3:28:14 PM

Creating an Independent (not part of a SPC chart) Frequency Histogram

The **FrequencyHistogramChart** class creates a standalone frequency histogram. It is a simple template where you need only supply data and set a few properties to create a proper frequency histogram. The example below extracted from the FrequencyHistogram.FrequencyHistogramApplication1 example program.

```
public class FrequencyApplication1 extends FrequencyHistogramChart {


      public FrequencyApplication1() {
.
.

      }

        private void InitializeGraph()
        {
```

```
        // Frequency bins
    double [] freqLimits = {19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5};
        // data to be sorted into frequency bins
        double [] freqValues = {32,44,44,42,57,
                                 26,51,23,33,27,
                                 42,46,43,45,44,
                                 53,37,25,38,44,
                                 36,40,36,48,56,
                                 47,40,58,45,38,
                                 32,39,43,31,45,
                                 41,37,31,39,33,
                                 20,50,33,50,51,
                                 28,51,40,52,43};

        // Initialize histogram
        this.initFrequencyHistogram(freqLimits, freqValues);
        // Set bar orientation
      this.getMainTitle().setTextString("Frequency Histogram of Selected Data");
        // Build chart
        this.setChartOrientation(ChartObj.VERT_DIR);
        this.setBarFillColor(ChartColors.LIGHTYELLOW);
      this.getFrequencyHistogramPlot().setSegmentFillColor(
          4,new Color(127, 127, 255));
        this.buildChart();
      }
.
.
.
}
```

All you have to do is supply the raw data, and the values of the frequency bins for which you want to accumulate values. The **FrequencyHistogramChart** class auto-scale a coordinate system, creates the proper x- and y-axes, and draws the resulting frequency histogram as a bar plot.

FrequencyHistogramChart.initFrequencyHistogram Method

Initializes the histogram frequency bin limits, and the data values for the histogram.

public void initFrequencyHistogram(
  double[] *frequencylimits*,

```
  double[] frequencyvalues
);
```

Parameters

frequencylimits

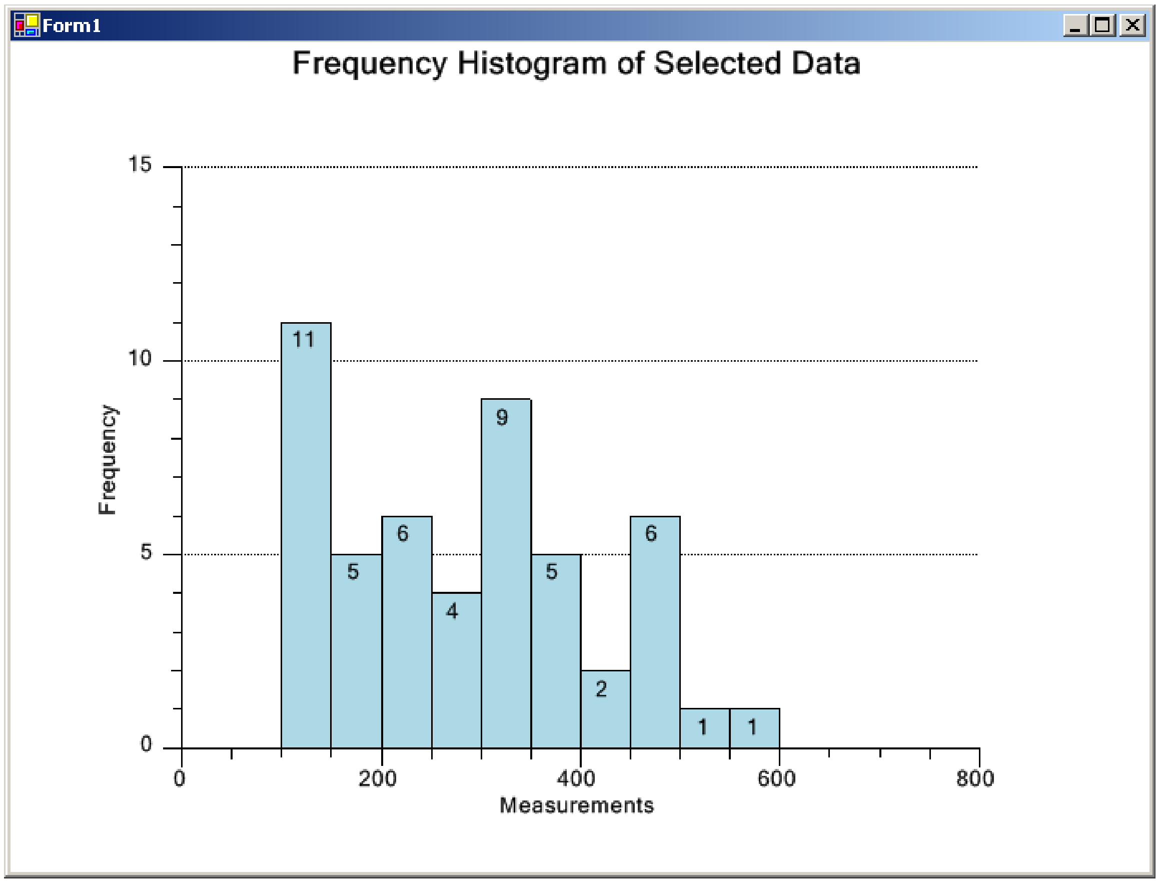
The frequency limits of the histogram bins.

frequencyvalues

An array the values that are counted with respect to the frequency bins.

The image below uses the following data:

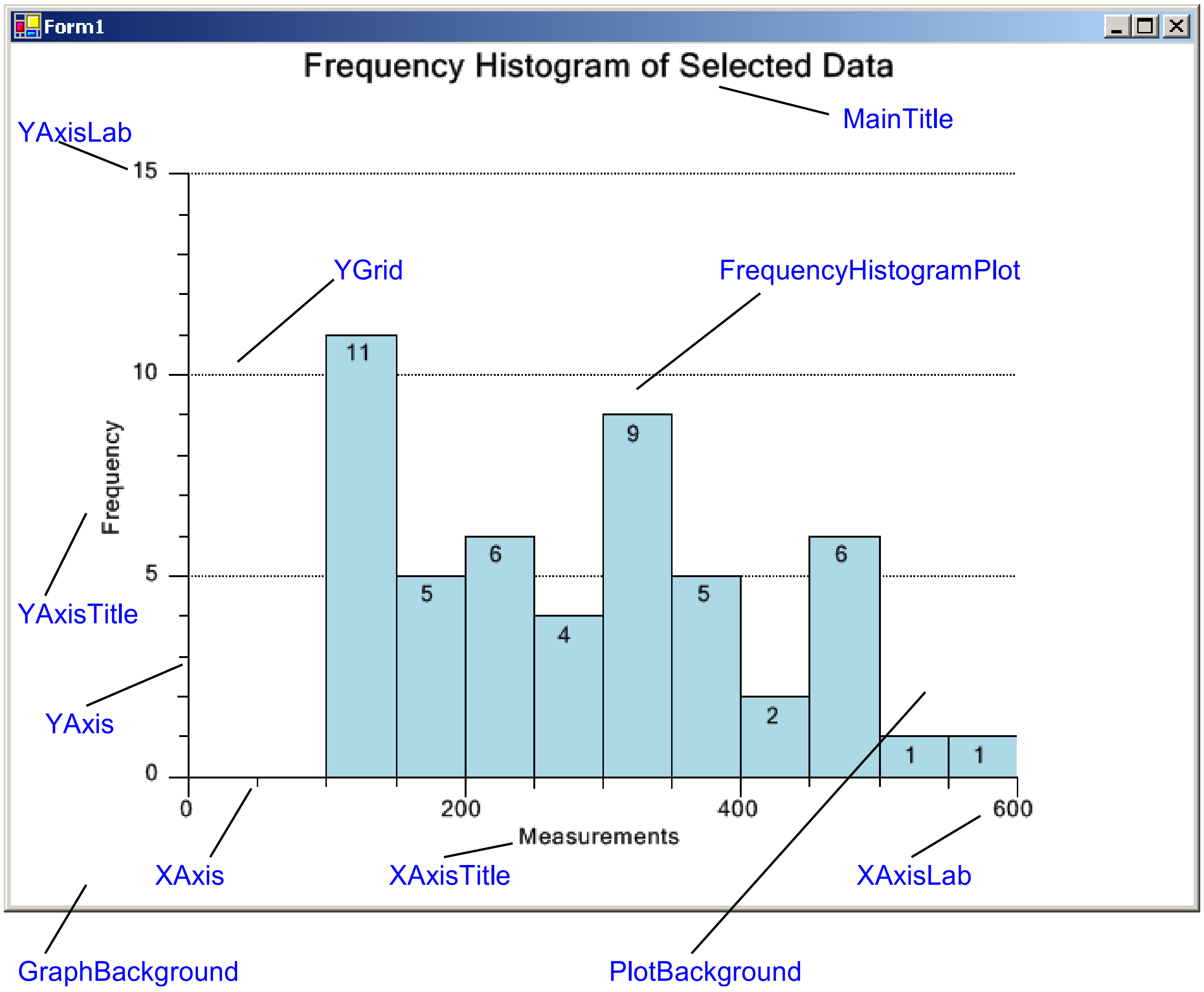
```
// Frequency bins
double [] freqLimits = {100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600};
// data to be sorted into frequency bins
double [] freqValues =    {121, 349, 345, 322, 277,
                          162, 218, 134, 133, 476,
                          323, 367, 133, 354, 245,
                          434, 476, 352, 185, 144,
                          165, 105, 461, 386, 263,
                          476, 304, 180, 557, 482,
                          327, 293, 539, 318, 251,
                          218, 472, 218, 199, 330,
                          109, 101, 137, 300, 119,
                          380, 410, 206, 122, 238};
```



Once the **init** routine is called, the chart can be further customized using the properties and methods below.
Class FrequencyHistogramChart – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/FrequencyHistogramChart.html

a link to the JavaDoc documentation for the class **FrequencyHistogramChart**

Changing Default Characteristics of the Chart



A **FrequencyHistogramChart** object has one distinct graph with its own set of properties. Once the graph is initialized (using the **initFrequencyHistogram**, or one of the **FrequencyHistogramChart** constructors**)**, you can modify the default characteristics of each graph using these properties. For example, you can change the color of y-axis, and the y-axis labels using the LineColor property of those objects.

```
void InitializeChart()
{
    // Frequency bins
double [] freqLimits = {100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600};
  // data to be sorted into frequency bins
    double [] freqValues =    {121, 349, 345, 322, 277,
                             162, 218, 134, 133, 476,
                             323, 367, 133, 354, 245,
                             434, 476, 352, 185, 144,
                             165, 105, 461, 386, 263,
                             476, 304, 180, 557, 482,
                             327, 293, 539, 318, 251,
                             218, 472, 218, 199, 330,
                             109, 101, 137, 300, 119,
// Initialize histogram
    this.initFrequencyHistogram(freqLimits, freqValues);
    this.getYAxis().setLineColor(ChartColors.GREEN);
    this.getYAxis().setLineWidth(3);
    this.getYAxisLab().setLineColor(ChartColors.DARKMAGENTA);
    this.buildChart();
}
```

Special Considerations

1. The **FrequencyHistogramChart** class uses the **QCChart2D HistogramPlot** plot object class to draw the histogram bars. That class uses individually assignable colors for each bar of the bar plot. The standard **ChartObj.setLineColor** and **ChartObj.setFillColor** methods do not work to change the color of the histogram bars in this case. Instead, you can change the histogram bar colors by calling **setSegmentLineColor** and **setSegmentFillColor**.

```
For (int i=0; i < 9; i++}
{
  this.getFrequencyHistogramPlot().setSegmentFillColor(i,ChartColors.BLUE);
  this.getFrequencyHistogramPlot().setSegmentLineColor(i,ChartColors.BLACK);
}
```

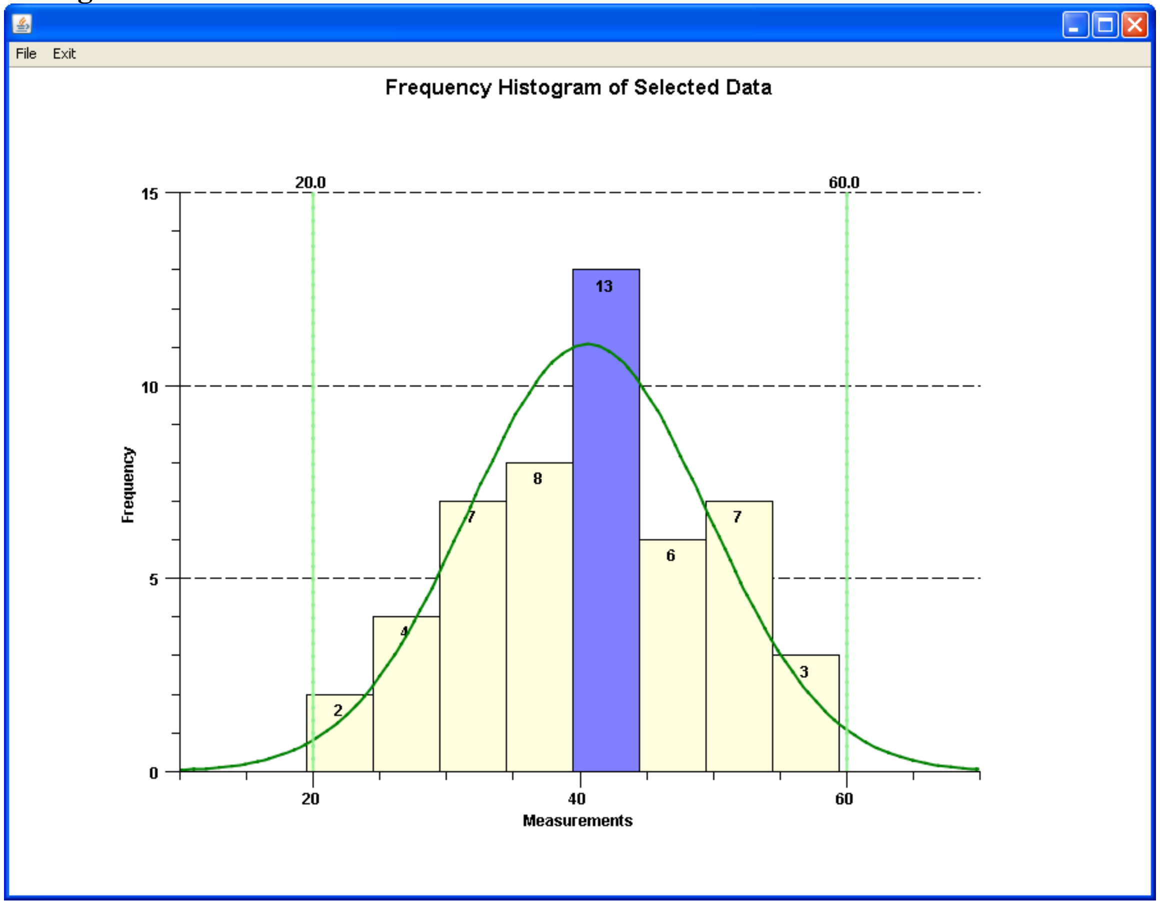
You can also use the utility method, **setBarFillColor**, **setBarLineColor** and **setBarLineWidth**, we added to the **FrequencyHistogramPlot** that will set all of the bars of the histogram plot at once.

```
  this.setBarFillColor(ChartColors.BLUE);
  this.setBarLineColor(ChartColors.BLACK);
```

Adding Control Lines and Normal Curve to Histogram Plot

Revision 1.7 adds a couple of new features to the histogram plot. First, you can add control limit alarm lines to the histogram plot. The control limit lines will be parallel to the frequency axis. Second, a normal distribution curve can be overlaid on top of the histogram data. The parameters are selected to give the normal distribution curve the same mean, standard deviation and area as the underlying histogram data. If the underlying data is normal, then there should be a relatively close fit between the normal curve and the underlying frequency data.

Histogram Control Limit Lines and Normal Curve fit



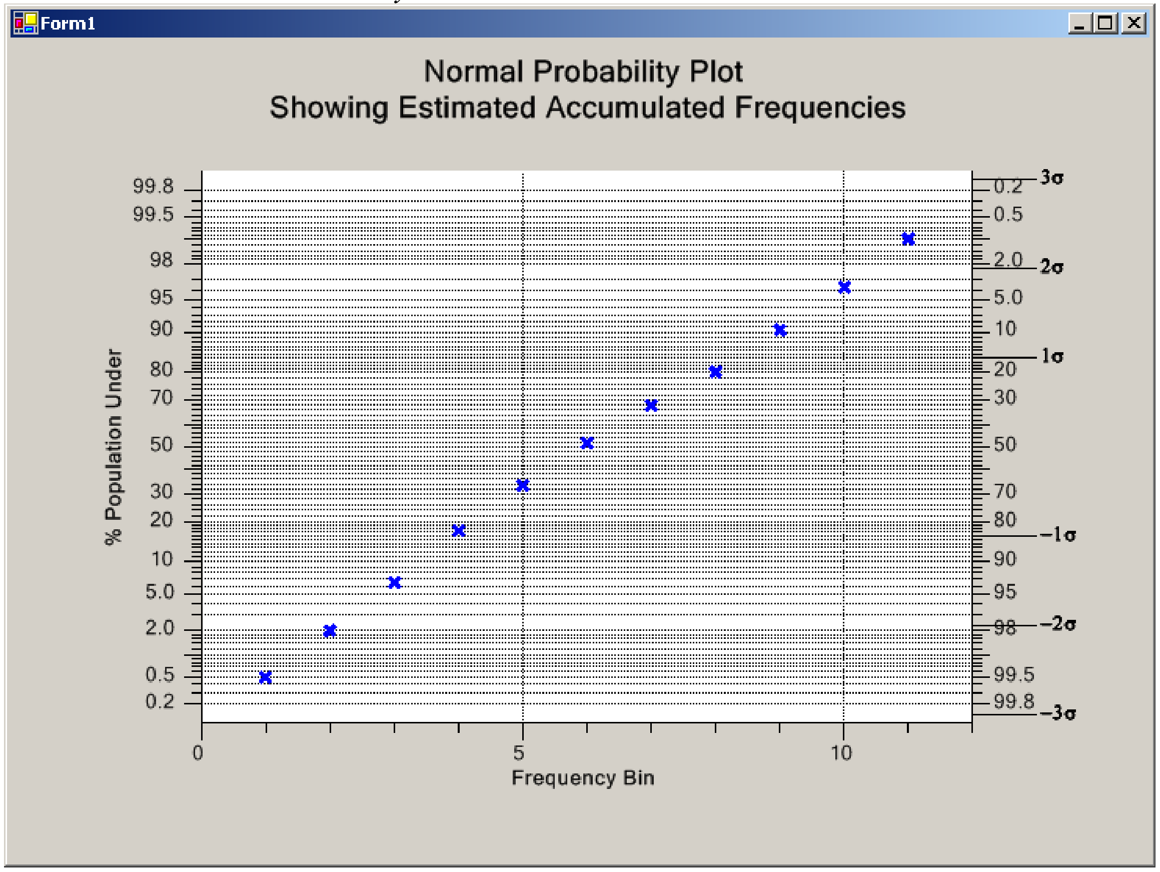
```
this.addFrequencyHistogramControlLine(20.0,
    new ChartAttribute(ChartColors.LIGHTGREEN, 2));
this.addFrequencyHistogramControlLine(60.0,
    new ChartAttribute(ChartColors.LIGHTGREEN, 2));
this.setAutoNormalCurve( true);
```

Probability Plots

Another important tool the SPC engineer uses to model the process variation is the probability plot. The probability plot is a simple way to test whether control chart measurements fit a normal distribution. Usually probability plot graphs are plotted by hand using special probability plot graph paper. We have added probability scale and axis classes that enable you to plot probability plots directly on the computer. Control chart measurements that follow a normal distribution curve will plot as a straight line when plotted in a normal probability plot.

Creating a Probability Plot

Cumulative Normal Probability Chart



The **ProbabilityChart** class creates a standalone probability plot. It is a simple template where you need only supply data and set a few properties to create a proper frequency histogram. The example below is extracted from the ProbabilityPlot.ProbabilityPlotApplication1 example program.

```
public class ProbabilityApplication1 extends ProbabilityChart {

      public ProbabilityApplication1() {
.
.
.
      }

        private void InitializeGraph()
        {

            // TODO: Add any initialization after the InitForm call
            double []x1 = {1.0,2,3,4,5,6,7,8,9};
            double []y1 = {2, 6, 13, 26, 58, 82, 93, 97,100};
            this.initProbabilityChart(x1, y1);
.
.
.
            this.buildChart();
        }   .
    .
    .
}
```

All you have to do is supply the raw data, and the values of the frequency bins for which you want to accumulate values. The **ProbabilityChart** class auto-scale a coordinate system, creates the proper x- and y-axes, and draws the resulting probability plot as a scatter plot.

ProbabilityChart.initProbabilityChart Method

Initializes the x- and y-values of the data points plotted in the probability plot.

public void initProbabilityChart(
  double[] *xvalues*,
  double[] *yvalues*
);

Parameters

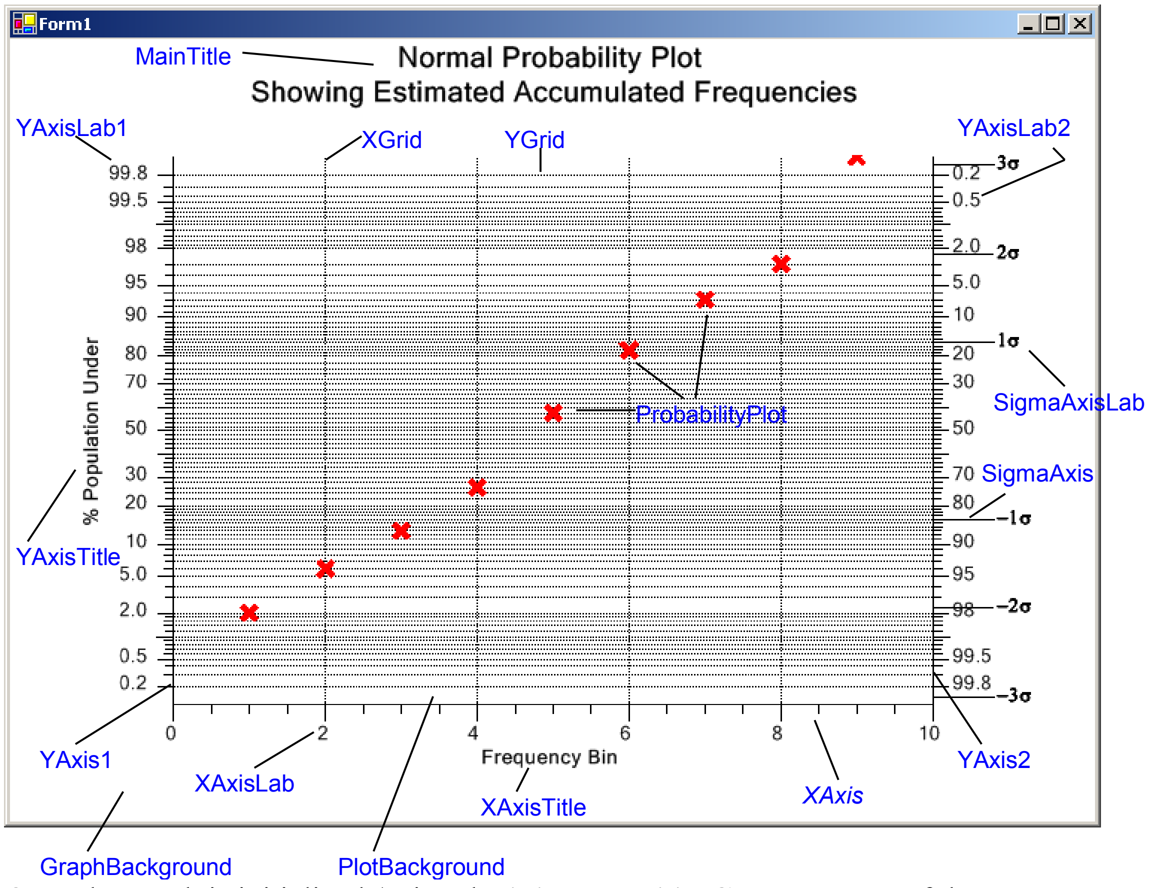
xvalues
    The x-values of the data points plotted in the probability plot.

yvalues
    The y-values of the data points plotted in the probability plot.

Class ProbabilityChart – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/ProbabilityChart.html

a link to the JavaDoc documentation for the class **ProbabilityChart**

Changing Default Characteristics of the Chart



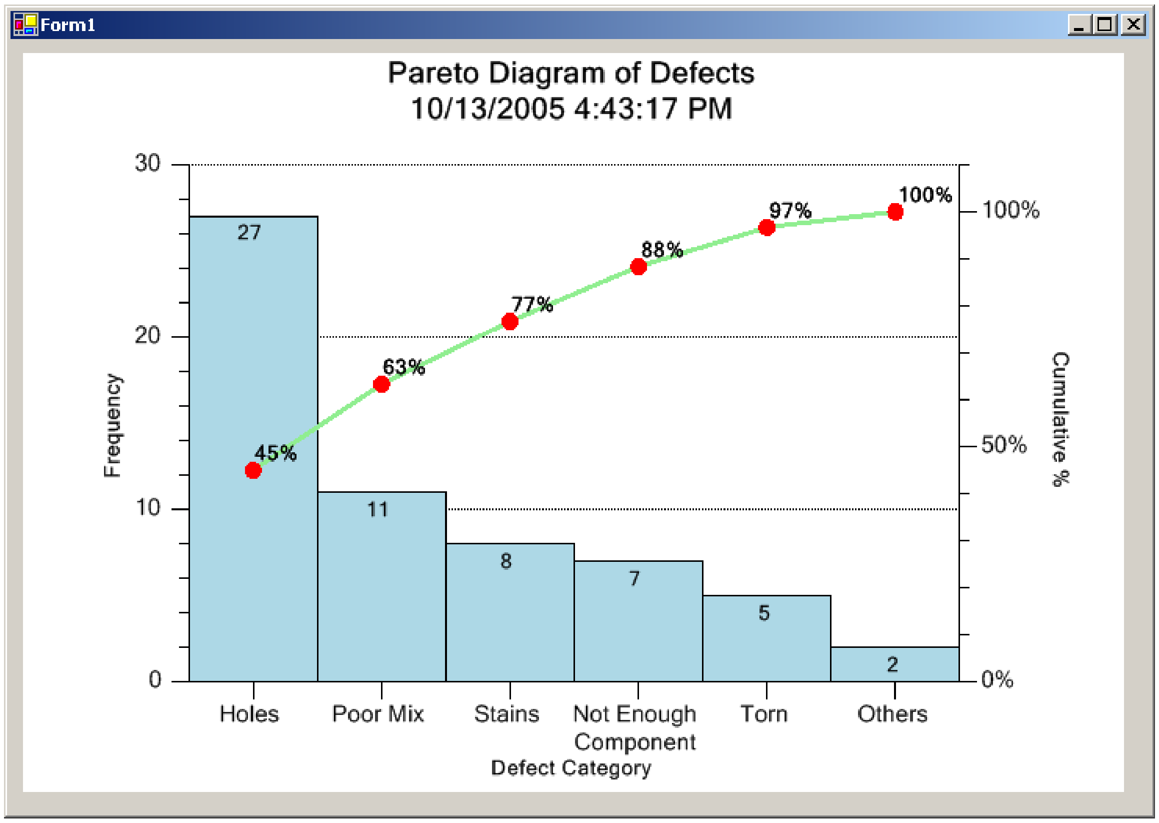
Once the graph is initialized (using the **initProbabilityChart** , or one of the **ProbabilityChart** constructors), you can modify the default characteristics of each graph using these properties. For example, you can change the color of y-axis, and the y-axis labels using the **LineColor** property of those objects.

```
void InitializeChart()
{
    // TODO: Add any initialization after the InitForm call
    double []x1 = {1,2,3,4,5,6,7,8,9};
    double []y1 = {2, 6, 13, 26, 58, 82, 93, 97,100};
    this.initProbabilityChart(x1, y1);

    this.getMainTitle().setTextString( "Normal Probability Plot");
    this.getMainTitle().addNewLineTextString("Showing Estimated Accumulated
Frequencies");

    this.getProbabilityPlot().setColor(ChartColors.RED);
    this.getProbabilityPlot().getChartObjAttributes().setSymbolSize(12);
    this.getYAxis1().setLineColor(ChartColors.GREEN);
    this.getYAxis1().setLineWidth( 3);
    this.getProbabilityPlot().setLineColor( ChartColors.RED);
    this.getProbabilityPlot().getChartObjAttributes().setSymbolSize(12);

    this.getYAxis2().setLineColor(ChartColors.BLUE);
    this.getYAxis2().setLineWidth( 3);
    this.getYAxisLab1().setLineColor( ChartColors.DARKMAGENTA);

    this.buildChart();
}
```

Pareto Diagrams

The Pareto diagram is special type of bar graph combined with a line plot, plotted in a chart that uses two different y-axis scales. The bar graph scale is a frequency scale that measures the number of times a specific problem has occurred. The line plot scale is a cumulative percentage scale.

Pareto Chart



The chart is easy to interpret. The tallest bar, the left-most one in a Pareto diagram, is the problem that has the most frequent occurrence. The shortest bar, the right-most one, is the problem that has the least frequent occurrence. Time spend on fixing the biggest problem will have the greatest affect on the overall problem rate. This is a simplistic view of actual Pareto analysis, which would usually take into account the cost effectiveness of fixing a specific problem. Never less, it is powerful communication tool that the SPC engineer can use in trying to identify and solve production problems.

Creating a Pareto Diagram

The **ParetoChart** class creates a standalone Pareto Diagram chart. It is a simple template where you need only supply data and set a few properties to create a proper frequency histogram. The example below is from the ParetoPlotApplication1 file of the ParetoDiagram example program.

```
public class ParetoApplication1 extends ParetoChart {



      public ParetoApplication1() {
.
.
.

      }

        private void InitializeGraph()
        {

            // add Pareto chart categories, values and strings
            this.addCategoryItem(5, "Torn");
            this.addCategoryItem(7, "Not Enough\nComponent");
            this.addCategoryItem(2, "Others");
            this.addCategoryItem(11, "Poor Mix");
            this.addCategoryItem(27, "Holes");
            this.addCategoryItem(8, "Stains");

            this.getMainTitle().setTextString("Pareto Diagram of Defects");

            this.buildChart();
        }
```

All you have to do is supply the raw data, and the values of the frequency bins for which you want to accumulate values. The **ParetoChart** class auto-scale a coordinate system, creates the proper x- and y-axes, and draws the resulting probability plot as a scatter plot.

ParetoChart.initParetoChart Method

Initializes the x- and y-values of the data points plotted in the probability plot.

```
public void initParetoChart (
  double[] categoryitems,
  string[] stringitems
);
```

Parameters

categoryitems
The values for each category in the Pareto chart.

stringitems
The strings identifying each category in the Pareto chart.

You can add the category item values and string item strings one by one using the addCategoryItem method.

ParetoChart.addCategoryItem Method

Add an item to the categoryValues and categoryStrings arrays.

```
public void addCategoryItem(
  double itemfreq,
  string itemstring
);
```

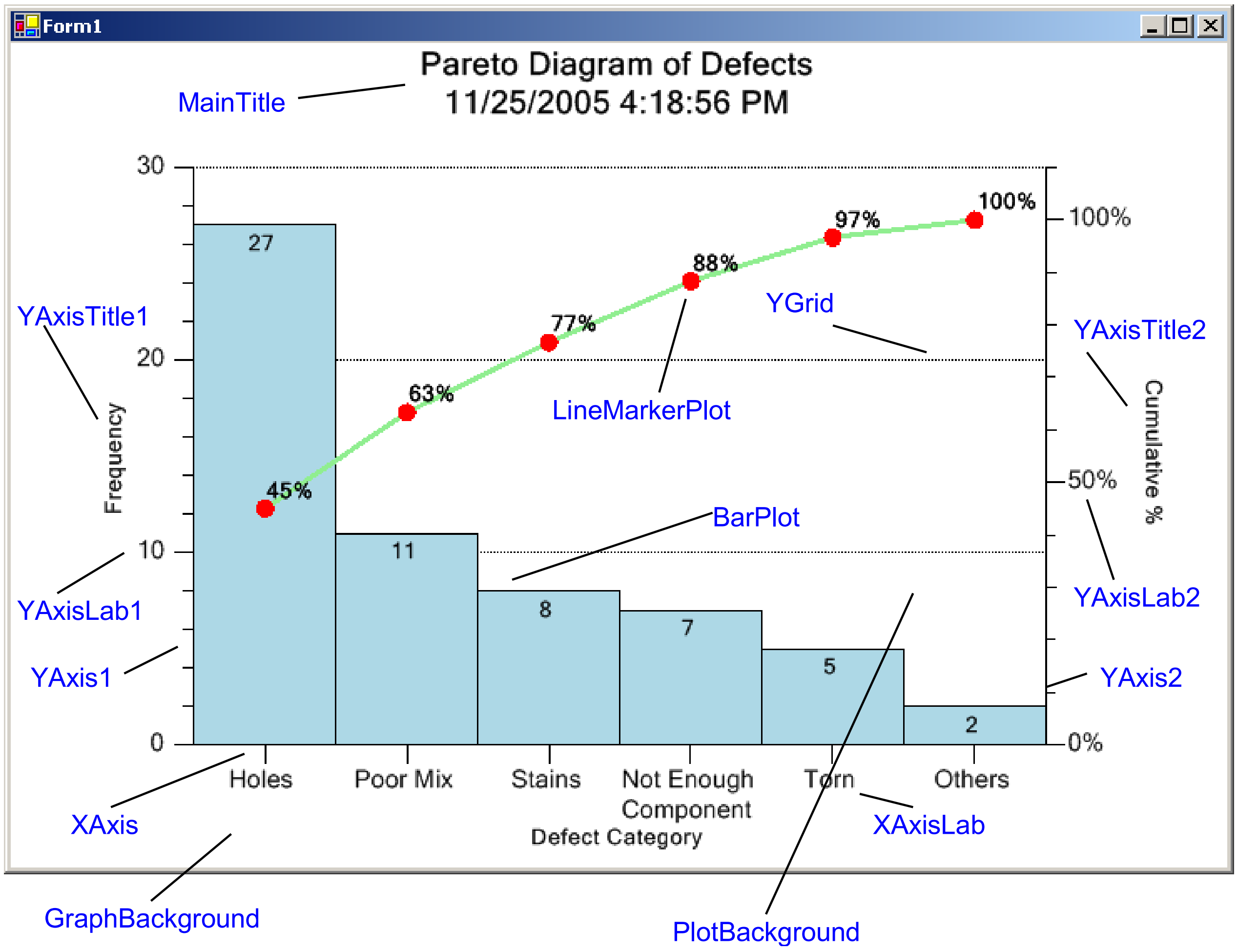
Parameters

itemfreq
The count of how many times this category has occurred.

itemstring
The string identifying the category item.

Class ParetoChart – http://www.quinn-curtis.com/qcspcchartjavahtml/com/quinncurtis/spcchartjava/ParetoChart.html a link to the JavaDoc documentation for the class **ParetoChart**

Changing Default Characteristics of the Chart



Once the graph is initialized (using the **initParetoChart** , or one of the **ParetoChart** constructors**)**, you can modify the default characteristics of each graph using these properties. For example, you can change the color of y-axis, and the y-axis labels using the **setLineColor** method of those objects.

```
private void InitializeGraph()
{

        // add Pareto chart categories, values and strings
        this.addCategoryItem(5, "Torn");
        this.addCategoryItem(7, "Not Enough\nComponent");
        this.addCategoryItem(2, "Others");
        this.addCategoryItem(11, "Poor Mix");
        this.addCategoryItem(27, "Holes");
        this.addCategoryItem(8, "Stains");
        this.getMainTitle().setTextString("Pareto Diagram of Defects");


        this.getLineMarkerPlot().setLineColor(ChartColors.GREEN);

        this.getLineMarkerPlot().getSymbolAttributes().setPrimaryColor(ChartColors.BLACK);
        this.getYAxis1().setLineColor(ChartColors.GREEN);
        this.getYAxis1().setLineWidth(3);
        this.getYAxisLab1().setLineColor(ChartColors.GREEN);

        this.getYAxis2().setLineColor(ChartColors.DARKMAGENTA);
        this.getYAxis2().setLineWidth(3);
        this.getYAxisLab2().setLineColor(ChartColors.DARKMAGENTA);

        this.buildChart();
}
```

10. File and Printer Rendering Classes

ChartPrint
BufferedImage

All of the chart classes described in this manual:

com.quinn-curtis.chart2djava.ChartView
- FrequencyHistogramChart
- ParetoChart
- ProbabilityChart
- SPCChartBase
  - SPCBatchAttributeControlChart
  - SPCBatchVariableControlChart
  - SPCTimeAttributeControlChart
  - SPCTimeVariableControlChart

are derived from the **com.quinncurtis.chart2djava.ChartView** class. The chart and table information displayed in these charts can be printed and saved to image files using the techniques described in the **QCChart2D** manual, QCChart2DJavaManual .pdf. This chapter repeats that information, substituting examples extracted from the for **SPC Control Chart Tools for Java** examples.

High quality B&W and color printing is an important feature of the charting library. The resulting graph renders on the printer using the resolution of the output device, for both text and graphical elements of the chart, and does not transfer a grainy image from the computer to the printer. The **QCChart2D for *Java*** software uses the Java **PrintDocument** component to implement printing. Options are included that allow different modes for positioning and sizing the chart on the printed page.

The **BufferedImage** class converts a chart into a Java **Bitmap** object, or saves the chart to a file in any of the graphics formats supported by the **System.Drawing.Imaging.ImageFormat** class. The image file is placeable in a web page or an application program.

Printing a Chart

Class ChartPrint

```
ChartObj
   |
   +--ChartPrint
```

The **ChartPrint** class implements the Java **Printable** interface to implement printing. The class selects, setups, and outputs a chart to a printer.

ChartPrint constructor

```
public ChartPrint(
   ChartView component,
   int nsizemode
);
```

component	Specifies the **ChartView** object to be printed.
nsizemode	Specifies the printer mapping mode. Use one of the mapping mode constants:

PRT_MAX	Print the view so that paper is used maximally. Text prints proportional to other objects, aspect ratio is maintained
PRT_EXACT	Print the view at the same size as the screen, at least as far as Java maintains a one to one correspondence in the printing engine. The aspect ratio of the view is maintained.
PRT_RECT	Print the view to the specified rectangle, specified using the setPrintRect method and normalized coordinates.

Call the **printDialog** method after creating the **ChartPrint** object. Then call the **startPrint** method, rendering the chart to the printer. If the **printDialog** method is not called prior to **startPrint**, the **startPrint** method automatically invokes the **printDialog** method. Subsequent calls to **startPrint** will not invoke the **printDialog** method.

ChartPrint example **(extracted from the example program FrequencyHistogramDemo.Frame1)**

```
  class MenuListener implements ActionListener {
    public void  actionPerformed(ActionEvent event)
    { Object object = event.getSource();

      if (object == printMenuItem)
      {
       frequencyapplication1.print(printgraph);
      } else
      if (object == printerSetupMenuItem)
      {
       printgraph.printDialog();
      } else
        if (object == imageSetupMenuItem)
        {
           frequencyapplication1.saveJPEG();
        } else
      if (object == exitMenuItem)
      {
        System.exit(0);;
      }
    }
   }

// From FrequencyApplication1
  public void print(ChartPrint printobj)
  {
    printobj.setPrintChartView(this);
    printobj.startPrint();
  }
```

Capturing the Chart as a Buffered Image

Class BufferedImage

ChartObj
```
   |
   +-- BufferedImage
```

The **ChartBufferedImage** class creates a **Bitmap** object that is used to render a **ChartView** object into an image buffer. The rendering takes place when the **ChartBufferedImage.render** method or **ChartBufferedImage.saveImage** method is called.

ChartBufferedImage constructor

```
public BufferedImage(
   ChartView component,
   int imgtype
);
```

```
public BufferedImage(
   ChartView component
);
```

component	The **ChartView** object that is the source for the chart image.
imagetype	The image type of the rendered image. Use one of the Java BufferedImage image type constants. BufferedImage.TYPE_INT_RGB is the default value.

The **ChartBufferedImage.getBufferedImage** method converts the chart to a Java **BufferedImage** object. If you want to save the image to a JPEG file, call the **ChartBufferedImage.saveImageAsJPEG** method

.

ChartBufferedImage example (extracted from the example program FrequencyHistogramDemo. FrequencyApplication1)

```
ChartView chartview = new ChartView();
.
.
.
public void SaveJPEG()
  { String filename = " FrequencyApplication1.jpg";
    ChartBufferedImage savegraphJPEG = new ChartBufferedImage();
    ImageFileChooser imagefilechooser = new ImageFileChooser(this);
    if (imagefilechooser.process(filename))
    {
    filename = imagefilechooser.getSelectedFilename();
    savegraphJPEG.setChartViewComponent(this);
    savegraphJPEG.render();
    savegraphJPEG.saveImageAsJPEG(filename);
    }
  }
```

11. Regionalization for non-USA English Markets

SPCChartStrings

Starting with Revision 3.0, we provide a much improved structure for adjusting the software for different cultures and languages. All of the pre-defined strings in the software have been moved to the static class SPCChartStrings, which can be modified at run-time, or, if you have the QCSPCChartJava source code, at compile time. You can create multiple sets of strings, one for each unique region you sell to, and initialize the software to that set at run-time. While something similar is often done using resource files in the final .Net application program, it was not possible to add a user-customizable resource file to a pre-compiled library, like our QCSPCChartJava.

Apart from the strings, there are a couple other areas which benefit from regionalization. First is the use of the "," (comma) in some locales as the decimal separator, in place of the "." (period). Our software uses the standard numeric conversion routines found in Java, for converting numeric values to strings, and these automatically take into account the proper format for the region recognized by the computer. So, you shouldn't have to do anything there.

Also, date/time values are subject to regional differences; specifically the order of the month-day-year in short form strings of the form 10/1/2011 ("M/dd/yyyy" US English format), compared to 1/10/2011 (("dd/M/yyyy" European format). There a couple of date/time formats in the list below, located at the enumerated values **defaultDateFormat** and **defaultTimeStampFormat.** You should change those to whatever time format you want.

The **SPChartStrings** class looks like:

```
public  enum   SPCStringEnum
{
    start,
    chartFont,
    highAlarmStatus,
    lowAlarmStatus,
    shortStringNo,
         shortStringYes,
  .
.
.
         end
     };

     // US-en
     static String[] usEnglishStrings = {
         "start", // used to mark the beginning of the array
         "Microsoft Sans Serif",  // default font string
         "H",          // alarm status line - High short string
         "L",          // alarm status line - Low short string
         "N",          // No short string
         "Y",          // Yes short string
.
.
.
         "end"  // used to mark the end of the array
   };

      static String[] currentDefaultStrings = usEnglishStrings;
}
```

The **SPCStringEnum** enumeration contains an item for every default string used in the software. Paired with that is the currentDefaultStringsList, which has one string element for every enumerated value in the **SPCStringEnum** enumeration. The software pulls the strings it uses from the currentDefaultStrings array. It is initialize by default with the usEnglishStrings array; the string values of which are listed in the tables below.

SPCStringEnum	**Default Strings**	**Description**
start=0	"start"	Marks the start of the enumeration
chartFont	"Microsoft Sans Serif"	default font string
highAlarmStatus	"H"	alarm status line - High short string
lowAlarmStatus	"L"	alarm status line - Low short string
shortStringNo	"N"	No short string
shortStringYes	"Y"	Yes short string

dataLogUserString	""	default data log user string
sPCControlChartDataTitle	"Variable Control Chart (X-Bar & R)"	Default chart title
zeroEqualsZero	"zero"	table zero string
timeValueRowHeader	"TIME"	TIME row header
alarmStatusValueRowHeader	"ALARM"	ALARM row header
numberSamplesValueRowHeader	"NO. INSP."	NO. INSP. row header
titleHeader	"Title: "	Title field caption
partNumberHeader	"Part No.: "	Part number field caption
chartNumberHeader	"Chart No.: "	Chart number field caption
partNameHeader	"Part Name: "	Part name field caption
operationHeader	"Operation:"	Operation field caption
operatorHeader	"Operator:"	Operator field caption
machineHeader	"Machine: "	Machine field caption
dateHeader	"Date: "	Date field caption
specificationLimitsHeader	"Spec. Limits: "	Spec limits field caption
gageHeader	"Gage: "	Chart number field caption
unitOfMeasureHeader	"Units: "	Chart number field caption
zeroEqualsHeader	"Zero Equals: "	Chart number field caption
defaultMean	"MEAN"	MEAN Calculated value row header
defaultMedian	"MEDIAN"	MEDIAN Calculated value row header
defaultRange	"RANGE"	RANGE Calculated value row header
defaultVariance	"VARIANCE"	VARIANCE Calculated value row header
defaultSigma	"SIGMA"	SIGMA Calculated value row header
defaultSum	"SUM"	SUM Calculated value row header

defaultSampleValue	"SAMPLE VALUE"	SAMPLE VALUE alculated value row header
defaultAbsRange	"ABS(RANGE)"	ABS(RANGE) Calculated value row header
defaultMovingAverage	"“MA”	
defaultCusumCPlus	""C+"	
defaultCusumCMinus	""C-"	
defaultEWMA	""EWMA"	
defaultPercentDefective	" "% DEF."	
defaultFractionDefective	""FRACT. DEF."	
defaultNumberDefective	""NO. DEF."	
defaultNumberDefects	""NO. DEF."	
defaultNumberDefectsPerUnit	""NO. DEF./UNIT"	
defaultNumberDefectsPerMillion	""DEF./MILLION"	
defaultPBar	""PBAR"	
defaultAttributeLCL	""LCLP"	
defaultAttributeUCL	""UCLP"	
defaultAbsMovingRange	"MR"	Moving Range Calculated value row header
defaultAbsMovingSigma	"MS"	Moving Sigam Calculated value row header
defaultX	"X"	Default string used to label centerline value of I-R chart.
defaultXBar	"XBAR"	Default string used to label centerline value for XBar chart
defaultRBar	"RBAR"	Default string used to label centerline value for Range chart
defaultTarget	"Target"	Default string used for target
defaultLowControlLimit	"LCL"	Default string used to label low control limit line

defaultLowAlarmMessage	"Low Alarm"	Default string used for low alarm limit message
defaultUpperControlLimit	"UCL"	Default string used to label high control limit line
defaultHighAlarmMessage	"High Alarm"	Default string used for high alarm limit message
defaultSampleRowHeaderPrefix	"Sample #"	Row header for Sample # rows
defaultDefectRowHeaderPrefix	"Defect #"	Row header for Defect # rows
batchColumnHead	"Batch #"	Default string used as the batch number column head in the log file.
timeStampColumn	"Time Stamp"	Default string used as the time stamp column head in the log file.
sampleValueColumn	"Sample #"	Default string used as the sample value column head in the log file.
notesColumn	"Notes"	Default string used as the notes value column head in the log file.
defaultDateFormat	"M/dd/yyyy"	Default date format used by the software.
defaultTimeStampFormat	"M/dd/yyyy HH:mm:ss"	Default full date/time format used by the software.
defaultDataLogFilenameRoot	"SPCDataLog"	Root string used for auto-naming of log data file.
dataLogFilename	"SPCDataLog"	Datalog default file name
frequencyHistogramXAxisTitle	"Measurements"	Frequency Histogram default x-axis title.
frequencyHistogramYAxisTitle	"Frequency"	Frequency Histogram default y-axis title.
frequencyHistogramMainTitle	"Frequency Histogram"	Frequency Histogram default main title.
paretoChartXAxisTitle	"Defect Category"	Pareto chart x-axis title
paretoChartYAxis1Title	"Frequency"	Pareto chart left y-axis title
paretoChartYAxis2Title	"Cumulative %"	Pareto chart right y-axis title
paretoChartMainTitle	"Pareto Diagram"	Pareto chart main title
probabilityChartXAxisTitle	"Frequency Bin"	Probability chart x-axis title
probabilityChartYAxisTitle	"% Population	Probability chart y-axis title

	Under"	
probabilityChartMainTitle	"Normal Probability Plot"	Probability chart main title
basic	"Basic"	Basic rules string
weco	"WECO"	WECO rules string
wecowsupp	"WECO+SUPPLEMENTAL"	WECO and Supplemental Rules string
nelson	"Nelson"	Nelson rules string
aiag	"AIAG"	AIAG rules string
juran	"Juran"	Juran rules string
hughes	"Hughes"	Hughes rules string
gitlow	"Gitlow"	Gitlow rules string
duncan	"Duncan"	Duncan rules string
primarychart	"Primary chart"	Used in alarm messages to specify the Primary Chart variable chart is in alarm
secondarychart	"Secondary chart"	Used in alarm messages to specify the Secondary Chart variable chart is in alarm
greaterthan	"greater than"	Used in alarm messages to specify that a greater than alarm limit has been violated
lessthan	"less than"	Used in alarm messages to specify that a less than alarm limit has been violated
above	"above"	Used in alarm messages to specify that values above a limit
below	"below"	Used in alarm messages to specify that values below a limit
increasing	"increasing"	Used in alarm messages to specify that values are increasing
decreasing	"decreasing"	Used in alarm messages to specify that values are decreasing
trending	"trending"	Used in alarm messages to specify that values are trending
within	"within"	Used in alarm messages to specify that values are

		within certain limits
outside	"outside"	Used in alarm messages to specify that values are outside certain limits
alternating	"alternating"	Used in alarm messages to specify that values are alternating about a limit value
centerline	"center line"	Used in alarm messages to specify the center line of the chart
sigmaShort	"S"	alarm status line - sigma short string
beyondAlarmStatus	"B"	alarm status line - beyond short string
trendingAlarmStatus	"T"	alarm status line - trending short string
stratificationAlarmStatus	"S"	alarm status line - stratification short string
oscillationAlarmStatus	"O"	alarm status line - oscillation short string
rule	"Rule"	used in alarm messages for word "Rule"
violation	"violation"	used in alarm messages for word "violation"
sigma	"sigma"	used in alarm messages for word "sigma"
target	"Target"	used in alarm messages for word "Target"
ucl	"UCL"	used in alarm messages for to designate Upper Control Limit
lcl	"LCL"	used in alarm messages for to designate Lower Control Limit
defaultCp	"Cp"	used to label Cp row of table
defaultCpl	"Cpl"	used to label Cpl row of table
defaultCpu	"Cpu"	used to label Cpu row of table
defaultCpk	"Cpk"	used to label Cpk row of table
defaultCpm	"Cpm"	used to label Cpm row of table
defaultPp	"PPp"	used to label Pp row of table
defaultPl	"Ppl"	used to label Pl row of table

defaultPu	"Ppu"	used to label Pu row of table
defaultPpk	"Ppk"	used to label Ppk row of table
end	"end"	Marks the end of the enumeration

The enumerated values can be used to access and modify any specific string.

```
String oldstring = SPCChartStrings.setString(

     SPCChartStrings.SPCStringEnum.defaultMean,"AVERAGE");


oldstring = SPCChartStrings.setString(

     SPCChartStrings.SPCStringEnum.defaultDateFormat,"dd/M/yyyy");
```

Where the static **setString** method returns the previous value for the string.

The **SPCChartStrings** module is used to define default, and static strings, for the various **QCSPCChart** classes, when those classes are initialized. Trying to set the **SPCChartStrings** strings using **setString** after any of the SPC charts have been instantiated will not have the desired effect. Since the SPC charts classes are normally instantiated in the main panel file, you must change any strings before that intialization takes place. The best way to do that is to initialize the string in a static method in the main Application file. You will find an example in the TimeVariableControlCharts.Application1 file. Call the static method using an initialization of a static variable in the global variables section of the class. This will guarantee that the strings get initialized first. Since the SPCChartStrings class is static, you can call it anytime. It does not need instantiation.

```
  static boolean initStringsComplete = initStrings();


  static boolean initStrings()

  {

     String oldstring =
SPCChartStrings.setString(  SPCChartStrings.SPCStringEnum.defaultMean,"MEAN");

      // Euro standard

     // oldstring = SPCChartStrings.setString(

       //      SPCChartStrings.SPCStringEnum.defaultDateFormat, "dd/M/yyyy");

      // us-Eng standard

      oldstring = SPCChartStrings.setString(

               SPCChartStrings.SPCStringEnum.defaultDateFormat, "M/dd/yyyy");

      return true;

  }
```

You can change every string used in the software, line by line, using the technique above. Another way is to create a properly sized and initialized string array and change every string in the software with a single call to the

SPCChartStrings.setCurrentDefaultStrings method. You will find an example in the MiscBatchBasedControlCharts.Application1 example.

```
 static boolean initStringsFlag = initStrings();

static boolean initStrings()
{
    String[] usEnglishStrings = {
    "start", // used to mark the beginning of the array
    "Microsoft Sans Serif",  // default font string
    "H",          // alarm status line - High short string
    "L",          // alarm status line - Low short string
    "N",          // No short string
    "Y",          // Yes short string
    "",            // default data log user string
    "Variable Control Chart (X-Bar & R)", // Default chart title
    "zero",  // table zero string
    "TIME",  // TIME row header
    "ALARM",  // ALARM row header
    "NO. INSP.", // NO. INSP. row header
    "Title: ",   // Title field caption
    "Part No.: ", // Part number field caption
    "Chart No.: ", // Chart number field caption
    "Part Name: ", // Part name field caption
    "Operation:" , // Operation field caption
    "Operator:" ,  // Operator field caption
    "Machine: ",  // Machine field caption
    "Date: ",     // Date field caption
    "Spec. Limits: ", // Spec limits field caption
    "Gage: ", // Chart number field caption
    "Units: ", // Chart number field caption
    "Zero Equals: ", // Chart number field caption
    "MEAN", // MEAN Calculated value row header
    "MEDIAN", // MEDIAN Calculated value row header
    "RANGE", // RANGE Calculated value row header
    "VARIANCE" , // VARIANCE Calculated value row header
    "SIGMA", // SIGMA Calculated value row header
    "SUM", // SUM Calculated value row header
    "SAMPLE VALUE", // SAMPLE VALUE alculated value row header
    "ABS(RANGE)", // ABS(RANGE) Calculated value row header
    "MA", // Moving Average
    "C+", // CuSum Plus string
```

```
        "C-", // CuSum Minus string
        "EWMA", // EWMA string
        "% DEF.", // Percent Defective
        "FRACT. DEF.", // Fraction Defective
        "NO. DEF.", // Number Defective
        "NO. DEF.", // Number Defects
        "NO. DEF./UNIT", // Number Defects per Unit
        "DEF./MILLION", // Number Defects per Million
        "PBAR", // Target label for Attribute charts
        "LCLP", // Low limit label for Attribute charts
        "UCLP", // High limit label for Attribute charts
        "MR", // Moving Range Calculated value row header
        "MS", // Moving Sigam Calculated value row header
        "X",  //Default string used to label centerline value of I-R chart.
        "XBAR", //  Default string used to label centerline value for XBar chart
        "RBAR", //   Default string used to label centerline value for Range chart
        "Target", //   Default string used for target
        "LCL", //   Default string used to label low control limit line
        "Low Alarm", //   Default string used for low alarm limit message
        "UCL",   // Default string used to label high control limit line
        "High Alarm", //   Default string used for high alarm limit message
        "Sample #", // Row header for Sample # rows
        "Defect #", // Row header for Defect # rows
        "Batch #", // Default string used as the batch number column head in the log
file.
        "Time Stamp", // Default string used as the time stamp column head in the
log file.
        "Sample #", // Default string used as the sample value column head in the
log file.
        "Notes", // Default string used as the notes value column head in the log
file.
        "M/dd/yyyy", // Default date format used by the software.
        "M/dd/yyyy HH:mm:ss", // Default full date/time format used by the software.
        "SPCDataLog", // Root string used for auto-naming of log data file.
        "SPCDataLog", // Datalog default file name, usually over-ridden when data
log opened.
        "Measurements", // Frequency Histogram default x-axis title.
        "Frequency", // Frequency Histogram default y-axis title.
        "Frequency Histogram", // Frequency Histogram default main title.
        "Defect Category", // Pareto chart x-axis title
        "Frequency", // Pareto chart left y-axis title
        "Cumulative %", // Pareto chart right y-axis title
        "Pareto Diagram", // Pareto chart main title
        "Frequency Bin", // Probability chart x-axis title
        "% Population Under", // Probability chart y-axis title
```

```
      "Normal Probability Plot", // Probability chart main title
      "Basic", // Basic +- 3-sigma rules
      "WECO", // WECO rules string
      "WECO+SUPPLEMENTAL", // WECO + Supplemental rules string
      "Nelson", // Nelson rules string
      "AIAG", // AIAG rules string
      "Juran", // Juran rules string
      "Hughes", // Hughes rules string
      "Gitlow", // Gitlow rules string
      "Duncan",// Duncan rules string
      "Primary chart", // Used in alarm messages to specify the Primary Chart
variable chart is in alarm
      "Secondary chart", // Used in alarm messages to specify the Secondary Chart
variable chart is in alarm
      "greater than", // Used in alarm messages to specify that a greater than
alarm limit has been violated
      "less than",  // Used in alarm messages to specify that a less than alarm
limit has been violated
      "above", // Used in alarm messages to specify that values above a limit
      "below", // Used in alarm messages to specify that values below a limit
      "increasing", // Used in alarm messages to specify that values are
increasing
      "decreasing", // Used in alarm messages to specify that values are
decreasing
      "trending", // Used in alarm messages to specify that values are trending
      "within", // Used in alarm messages to specify that values are within
certain limits
      "outside", // Used in alarm messages to specify that values are outside
certain limits
      "alternating", // Used in alarm messages to specify that values are
alternating about a limit value
      "center line", // Used in alarm messages to specify the center line of the
chart
      "S",        // Used in alarm messages  as sigma short string
      "B",        // alarm status line - beyond short string
      "T",        // alarm status line - trending short string
      "S",        // alarm status line - stratification short string
      "O",        // alarm status line - oscillation short string
      "Rule", // used in alarm messages for word "Rule"
      "violation", // used in alarm messages for word "violation"
      "sigma", // used in alarm messages for word "sigma"
      "Target", // used in alarm messages for word "Target"
      "UCL", // used in alarm messages for to designate Upper Control Limit
      "LCL", // used in alarm messages for to designate Lower Control Limit
      "Cp", // Cp string
      "Cpl", // Cpl string
      "Cpu", // Cpu string
```

```
    "Cpk", // Cpk string
    "Cpm", // Cpm string
    "Pp",  // Pp string
    "Pl", // Pl string
    "Pu",  // Pu string
    "Ppk",  // Ppk string
    "end"};  // used to mark the end of the array

 SPCChartStrings.setCurrentDefaultStrings(usEnglishStrings);
 return true;
}
```

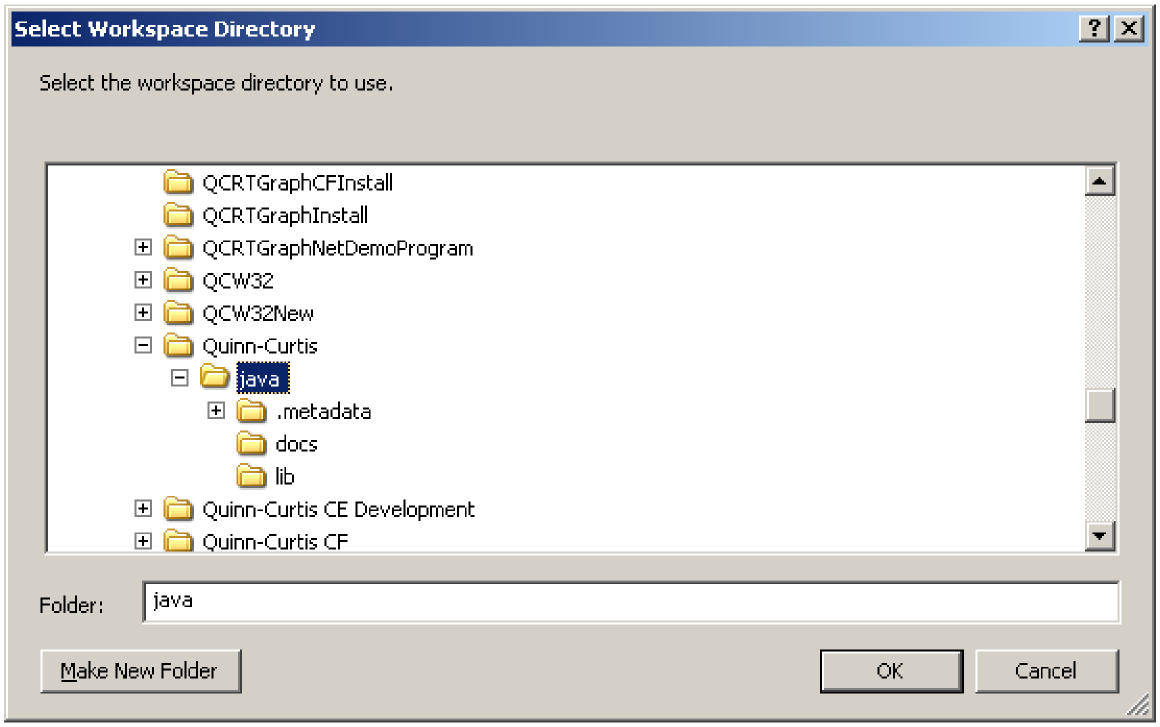
12. Compiling and Running the Example Programs

Using Eclipse, NetBeans and JBuilder IDE's

Eclipse

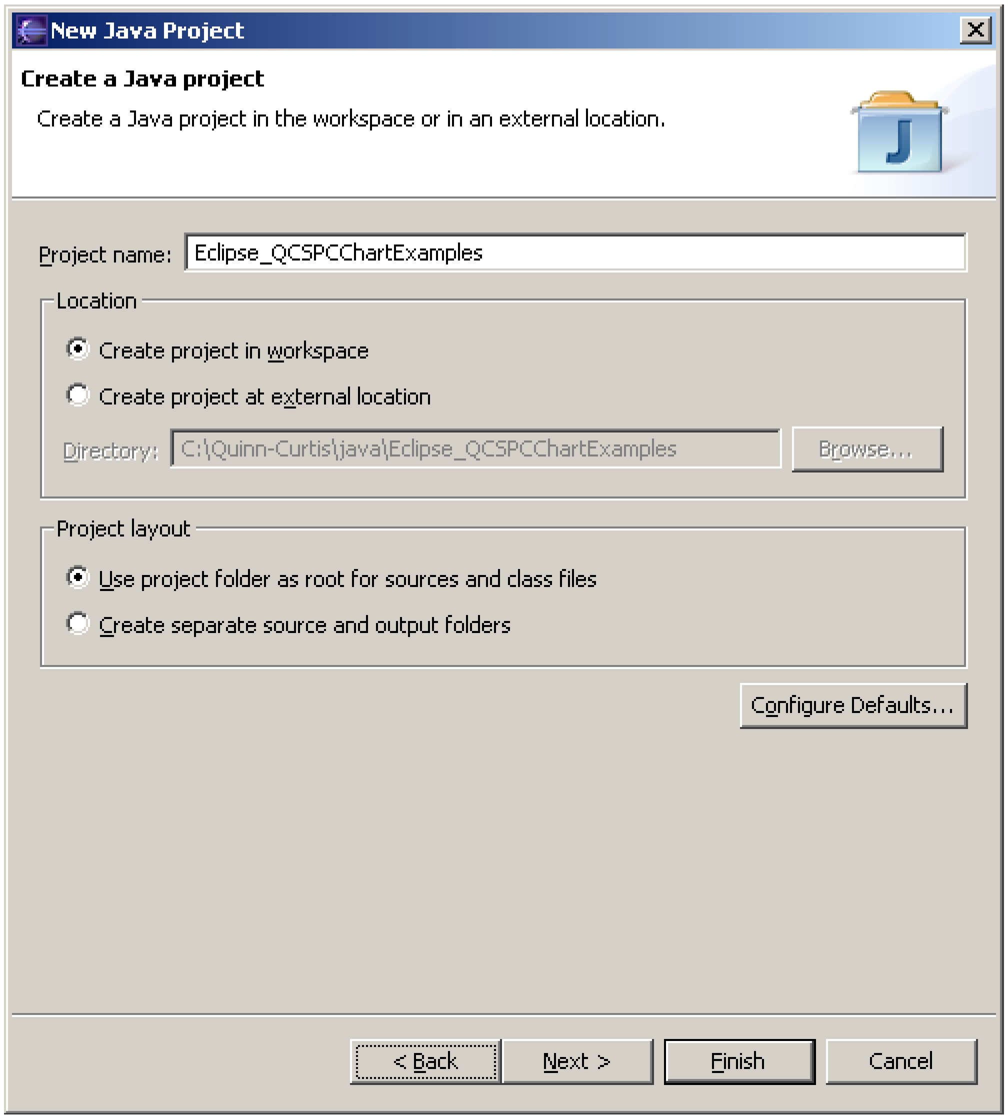
These directions were created using Eclipse Platform Version 3.0.2. Our standard development environment is Eclipse and our example programs are organized along the lines of the Java example programs that ship with Eclipse.

From the Eclipse Java Perspective IDE, select **File | Switch Workspace** and browse to and select the **C:\Quinn-Curtis\java** directory.



Folder: java

This will establish the current workspace directory as Quinn-Curtis/java. From the Eclipse main menu select **File | New | Project | Java Project.** Give the project the name Eclipse_QCSPCChartJavaExamples (any name will work).



Project name: Eclipse_QCSPCChartExamples

Location

(●) Create project in workspace

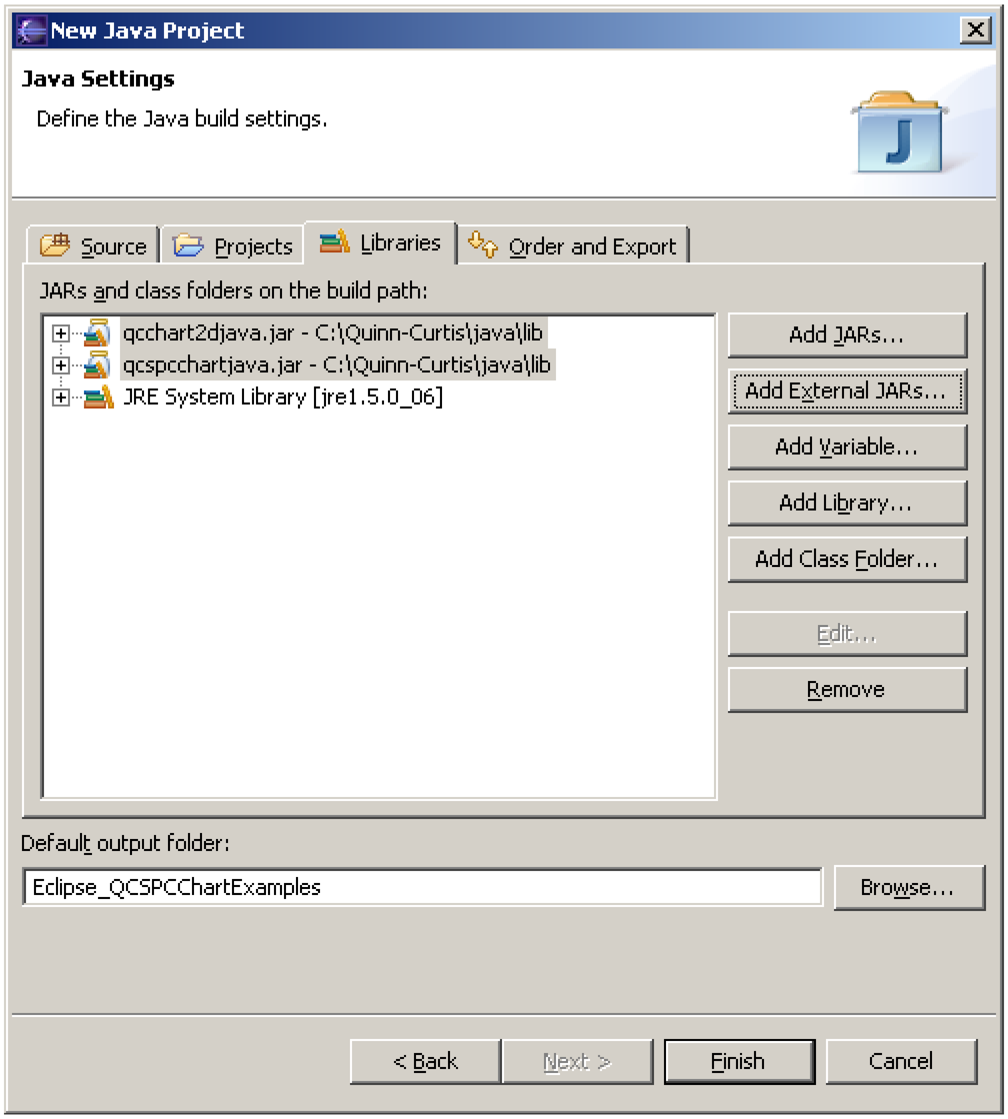
( ) Create project at external location

Directory: C:\Quinn-Curtis\java\Eclipse_QCSPCChartExamples

(●) Use project folder as root for sources and class files

( ) Create separate source and output folders

Select **Next** to define the Java build settings. Select the **Libraries** tab and select **Add External JARs** and browse to the Quinn-Curtis/java/lib directory and add the qcchart2djava.jar and qcspcchartjava.jar files.



JARs and class folders on the build path:

```
qcchart2djava.jar - C:\Quinn-Curtis\java\lib
qcspcchartjava.jar - C:\Quinn-Curtis\java\lib
JRE System Library [jre1.5.0_06]
```

Default output folder: Eclipse_QCSPCChartExamples

All of the example programs reference the QCCChart2D and QCSPCChart for Java jar files (qcchart2djava.jar and qcspcchartjava.jar) located in the Quinn-Curtis/java/lib folder. The qcchart2djava.jar file contains the **com.quinncurtis.chart2djava** package referenced in the

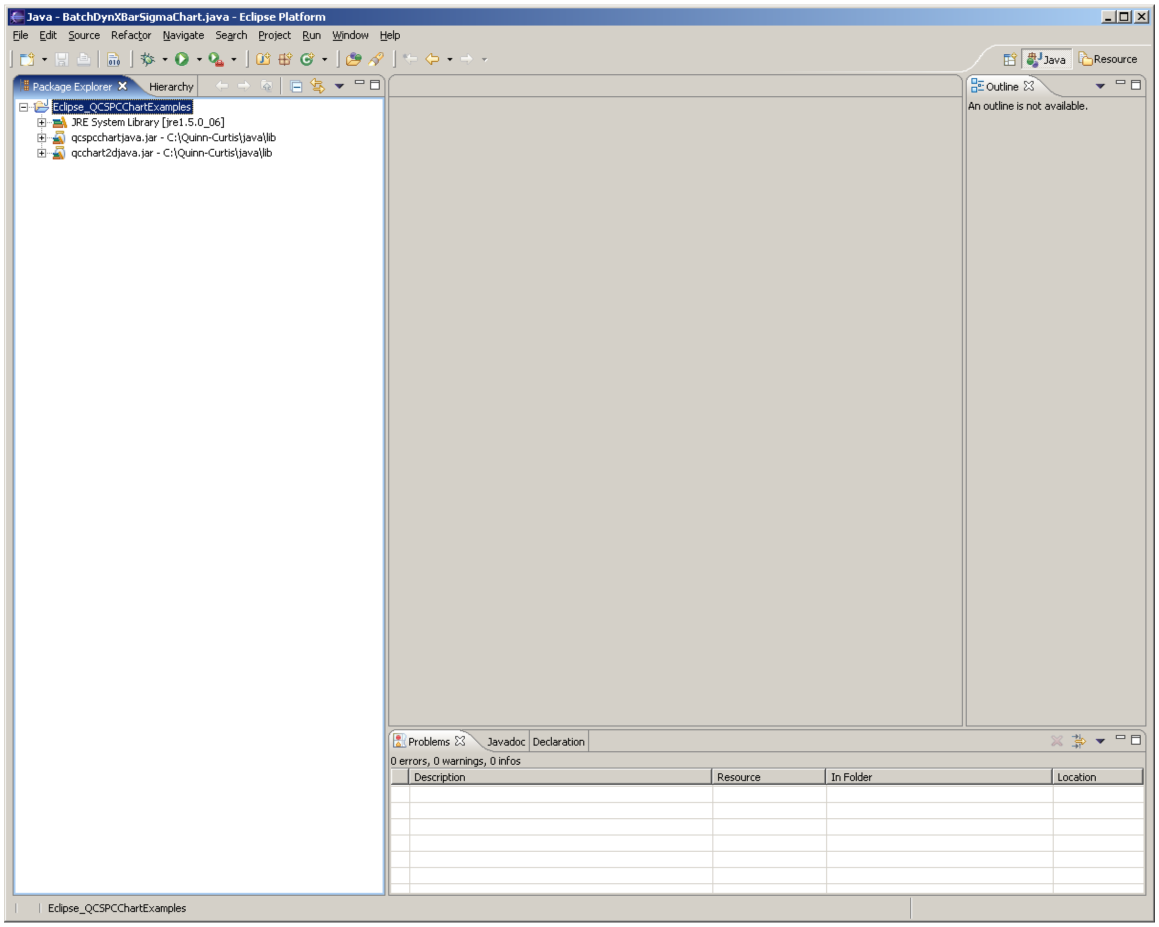
```
import com.quinncurtis.chart2djava.*;
```

statements found in all of the example program source files that contain calls to the charting routines.. The qcspcchartjava.jar file contains the **com.quinncurtis.spcchartjava** package referenced in the

```
import com.quinncurtis.spcchartjava.*;
```

statements found in all of the real-time graphics example program source files.

Select **Finish** to complete the creation of the basic project settings. Say Yes if it asks you for the Perspective Switch. The Package Explorer should look like:


Java - BatchDynXBarSigmaChart.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Resource
Package Explorer
Hierarchy
Outline
Problems
Javadoc
Declaration
Description
Resource
In Folder
Location

Eclipse_QCSPCChartExamples
JRE System Library [jre1.5.0_06]
qcspcchartjava.jar - C:\Quinn-Curtis\java\lib
qcchart2djava.jar - C:\Quinn-Curtis\java\lib

An outline is not available.

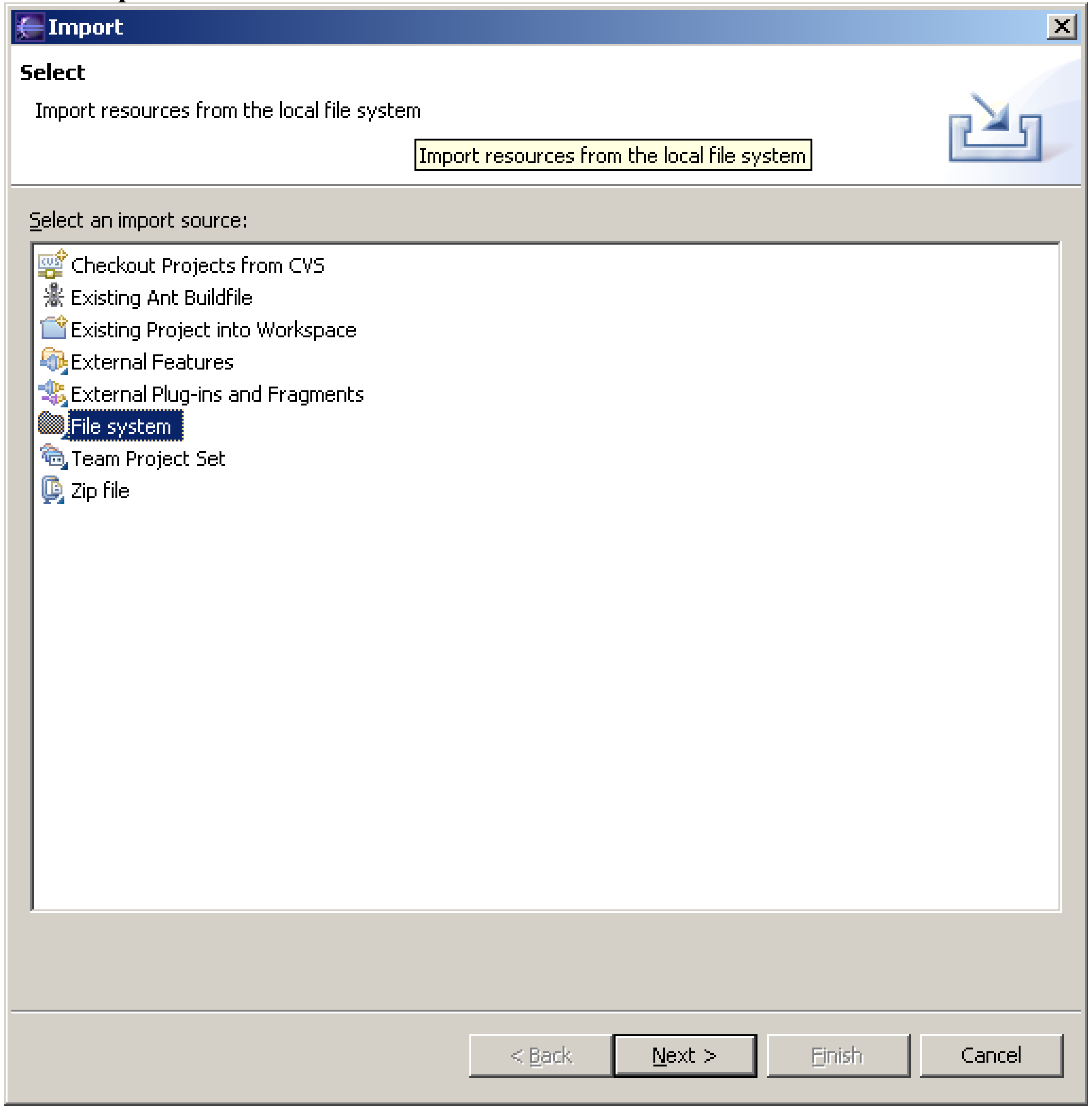
0 errors, 0 warnings, 0 infos

Eclipse_QCSPCChartExamples

To add the QCChart2D Javadoc documentation to the project: right click on the **qcchart2djava.jar** node and select **Properties** then **Javadoc Location**. Select the **Javadoc in Archive** radio button, and browse to qcchart2djava.jar file as the **Archive Path**, then select **doc** as the **Path within Archive**. From that point on Eclipse F1 and Shift-F2 help should be available on the qcchart2djava classes. **Repeat** the process for the **qcspcchartjava.jar** library file.

The project is still without the source files of the QCChart2D and QCSPCChart Java examples and these are added next.

Right click the Eclipse_QCSPCChartJavaExamples project in the Package Explorer and select **Import**.


Import

Select

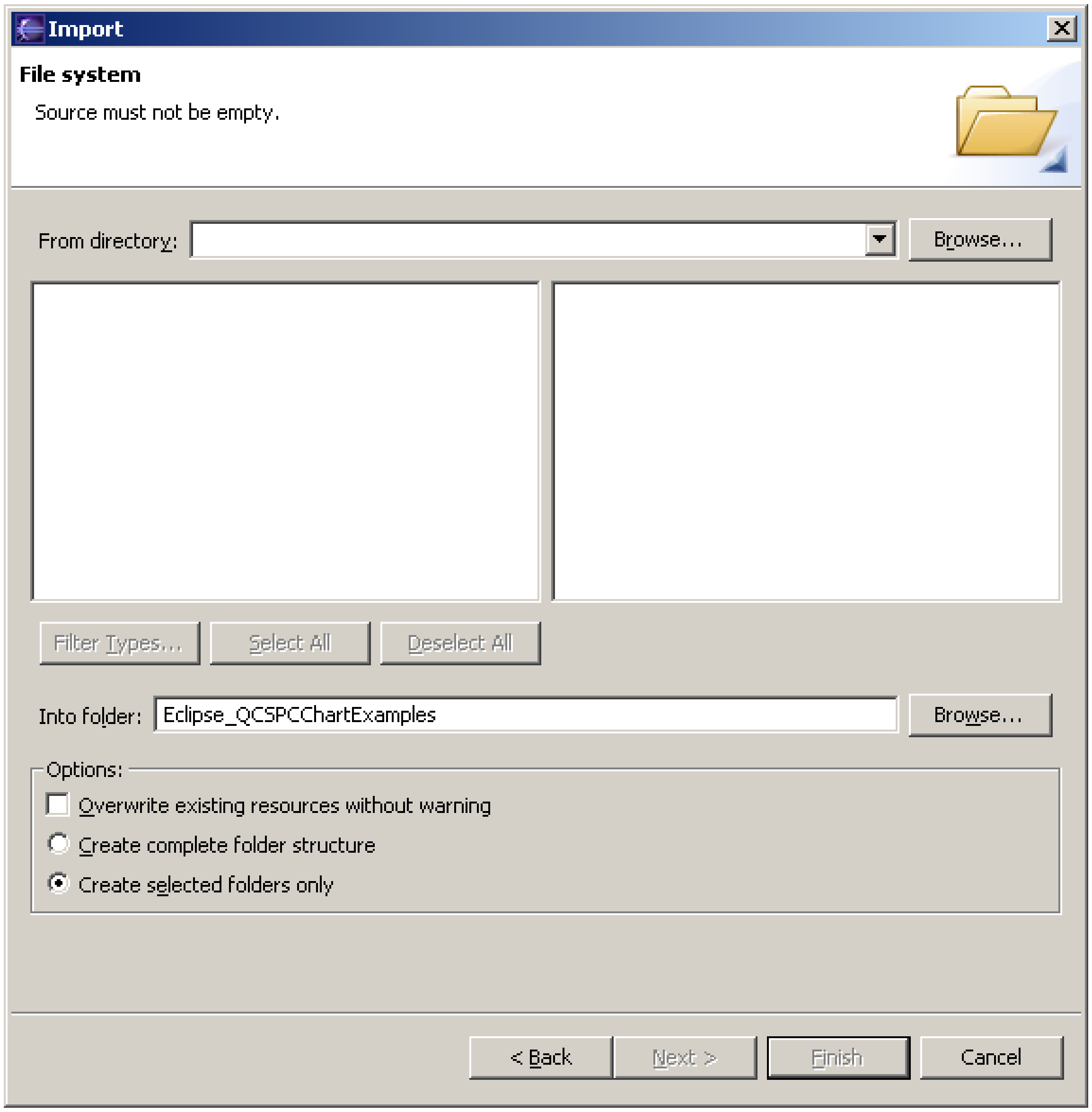
Import resources from the local file system

Import resources from the local file system

Select an import source:

- Checkout Projects from CVS
- Existing Ant Buildfile
- Existing Project into Workspace
- External Features
- External Plug-ins and Fragments
- File system
- Team Project Set
- Zip file

The Select **File System** and Browse to the Quinn-Curtis/java directory. Select Next.



File system

Source must not be empty.

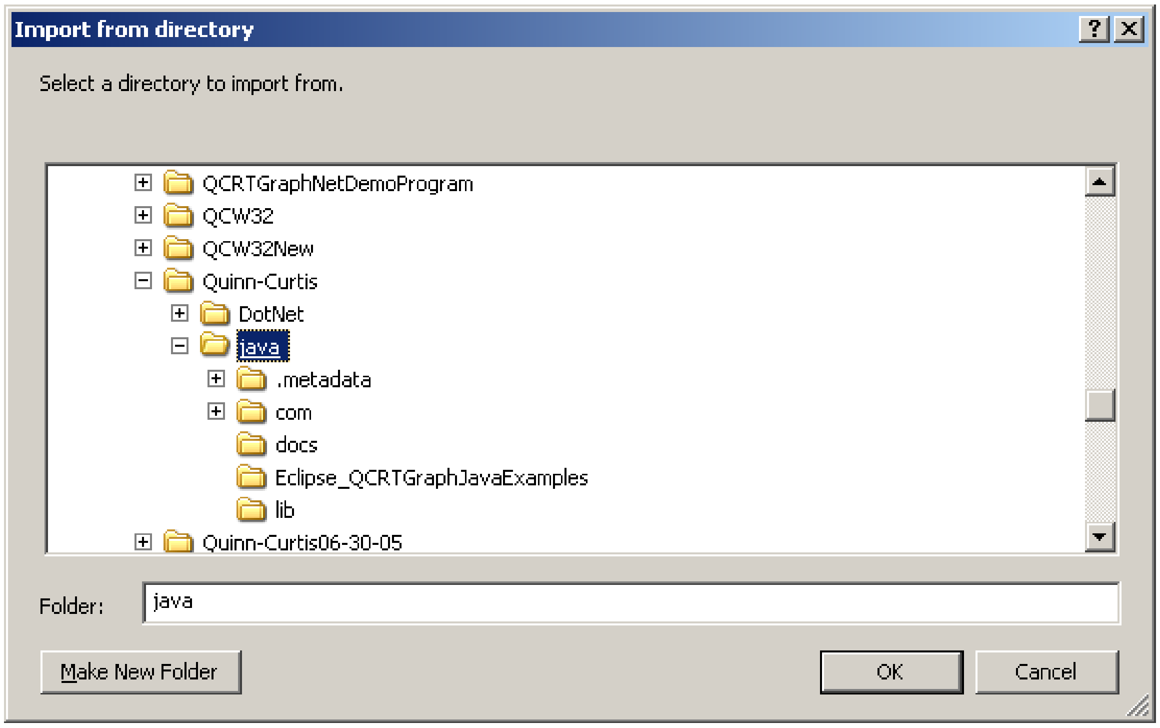
From directory:

Into folder: Eclipse_QCSPCChartExamples

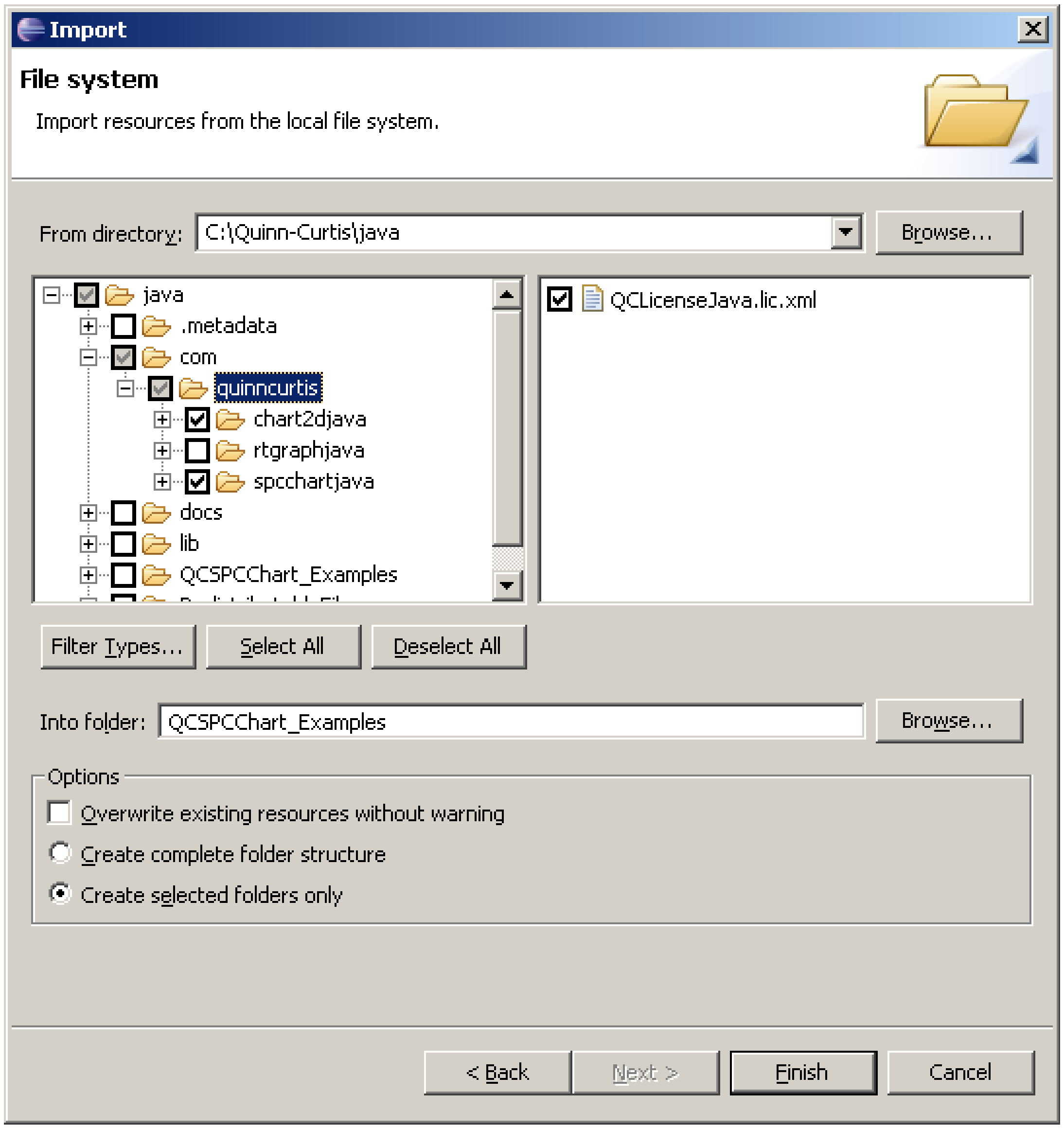
Options:

- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

The **Into Folder** field should say Eclipse_QCSPCChartJavaExamples. Select the **From Directory** Browse button and browse to the Quinn-Curtis\java folder


Import from directory
Select a directory to import from.
QCRTGraphNetDemoProgram
QCW32
QCW32New
Quinn-Curtis
DotNet
java
.metadata
com
docs
Eclipse_QCRTGraphJavaExamples
lib
Quinn-Curtis06-30-05
Folder:
java
Make New Folder
OK
Cancel

Select OK.


Import

File system

Import resources from the local file system.

From directory: C:\Quinn-Curtis\java

- [✓] java
  - [ ] .metadata
  - [✓] com
    - [✓] quinncurtis
      - [✓] chart2djava
      - [ ] rtgraphjava
      - [✓] spcchartjava
  - [ ] docs
  - [ ] lib
  - [ ] QCSPCChart_Examples

[✓] QCLicenseJava.lic.xml

Into folder: QCSPCChart_Examples

Options

[ ] Overwrite existing resources without warning

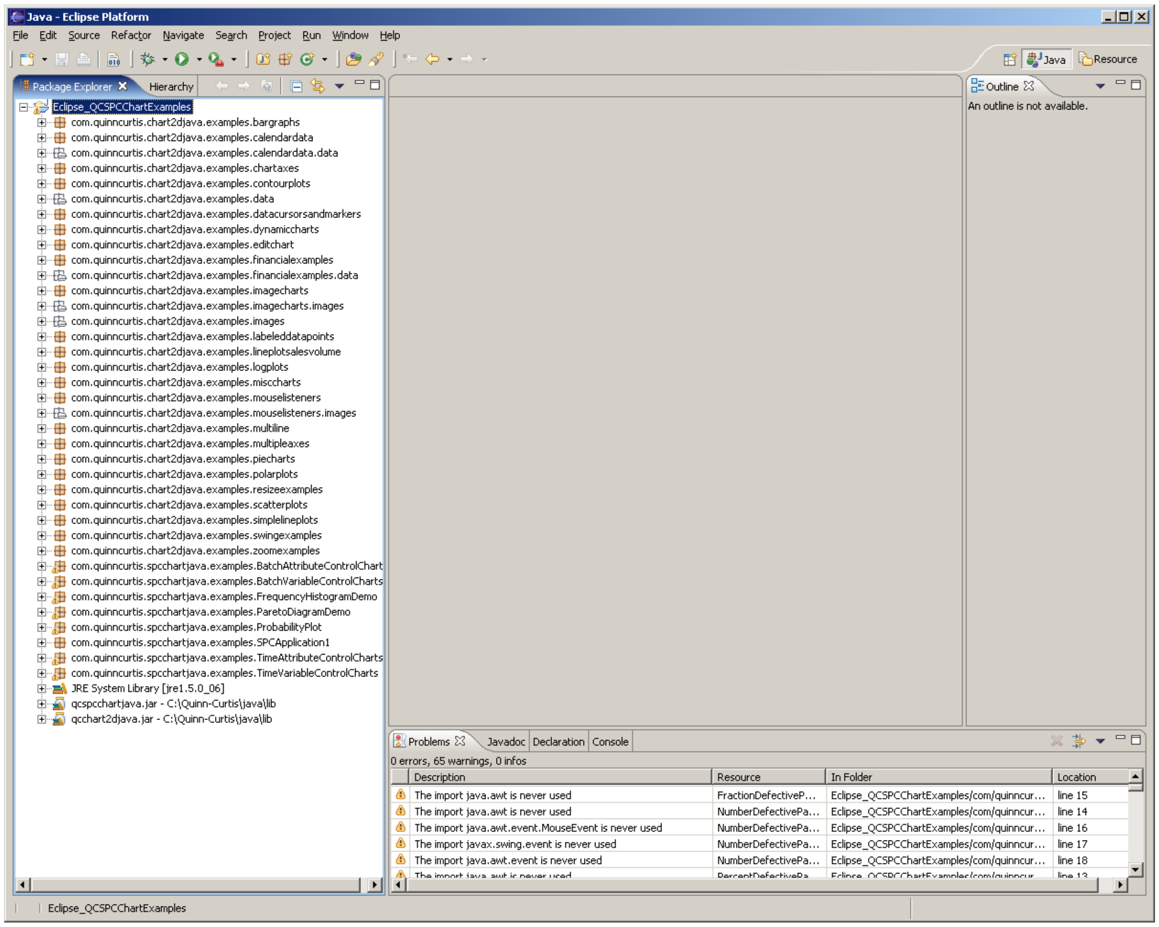
( ) Create complete folder structure

(●) Create selected folders only

Expand the Quinn-Curtis/java directory and check the chart2djava and spcchartjava folders in the com folder.

Select **Finish** and the example program folders under com/quinncurtis will be added to the project. The entire contents of the example program folders will be copied so com/quinncurtis/ directory structure is recreated under the Eclipse_QCSPCChartJavaExamples folder.

Expand one of the example program folders and locate the Application1.java file. Right click on that file and select **Run | Run Java Application**.



Eclipse_QCSPCChartExamples
- com.quinncurtis.chart2djava.examples.bargraphs
- com.quinncurtis.chart2djava.examples.calendardata
- com.quinncurtis.chart2djava.examples.calendardata.data
- com.quinncurtis.chart2djava.examples.chartaxes
- com.quinncurtis.chart2djava.examples.contourplots
- com.quinncurtis.chart2djava.examples.data
- com.quinncurtis.chart2djava.examples.datacursorsandmarkers
- com.quinncurtis.chart2djava.examples.dynamiccharts
- com.quinncurtis.chart2djava.examples.editchart
- com.quinncurtis.chart2djava.examples.financialexamples
- com.quinncurtis.chart2djava.examples.financialexamples.data
- com.quinncurtis.chart2djava.examples.imagecharts
- com.quinncurtis.chart2djava.examples.imagecharts.images
- com.quinncurtis.chart2djava.examples.images
- com.quinncurtis.chart2djava.examples.labeleddatapoints
- com.quinncurtis.chart2djava.examples.lineplotsalesvolume
- com.quinncurtis.chart2djava.examples.logplots
- com.quinncurtis.chart2djava.examples.misccharts
- com.quinncurtis.chart2djava.examples.mouselisteners
- com.quinncurtis.chart2djava.examples.mouselisteners.images
- com.quinncurtis.chart2djava.examples.multiline
- com.quinncurtis.chart2djava.examples.multipleaxes
- com.quinncurtis.chart2djava.examples.piecharts
- com.quinncurtis.chart2djava.examples.polarplots
- com.quinncurtis.chart2djava.examples.resizeexamples
- com.quinncurtis.chart2djava.examples.scatterplots
- com.quinncurtis.chart2djava.examples.simplelineplots
- com.quinncurtis.chart2djava.examples.swingexamples
- com.quinncurtis.chart2djava.examples.zoomexamples
- com.quinncurtis.spcchartjava.examples.BatchAttributeControlChart
- com.quinncurtis.spcchartjava.examples.BatchVariableControlCharts
- com.quinncurtis.spcchartjava.examples.FrequencyHistogramDemo
- com.quinncurtis.spcchartjava.examples.ParetoDiagramDemo
- com.quinncurtis.spcchartjava.examples.ProbabilityPlot
- com.quinncurtis.spcchartjava.examples.SPCApplication1
- com.quinncurtis.spcchartjava.examples.TimeAttributeControlCharts
- com.quinncurtis.spcchartjava.examples.TimeVariableControlCharts
- JRE System Library [jre1.5.0_06]
- qcspcchartjava.jar - C:\Quinn-Curtis\java\lib
- qcchart2djava.jar - C:\Quinn-Curtis\java\lib

An outline is not available.

Problems | Javadoc | Declaration | Console

0 errors, 65 warnings, 0 infos

	Description	Resource	In Folder	Location
⚠	The import java.awt is never used	FractionDefectiveP...	Eclipse_QCSPCChartExamples/com/quinncur...	line 15
⚠	The import java.awt is never used	NumberDefectivePa...	Eclipse_QCSPCChartExamples/com/quinncur...	line 14
⚠	The import java.awt.event.MouseEvent is never used	NumberDefectivePa...	Eclipse_QCSPCChartExamples/com/quinncur...	line 16
⚠	The import javax.swing.event is never used	NumberDefectivePa...	Eclipse_QCSPCChartExamples/com/quinncur...	line 17
⚠	The import java.awt.event is never used	NumberDefectivePa...	Eclipse_QCSPCChartExamples/com/quinncur...	line 18
⚠	The import java.awt is never used	PercentDefectivePa	Eclipse_QCSPCChartExamples/com/quinncur	line 13

Eclipse_QCSPCChartExamples

NetBeans

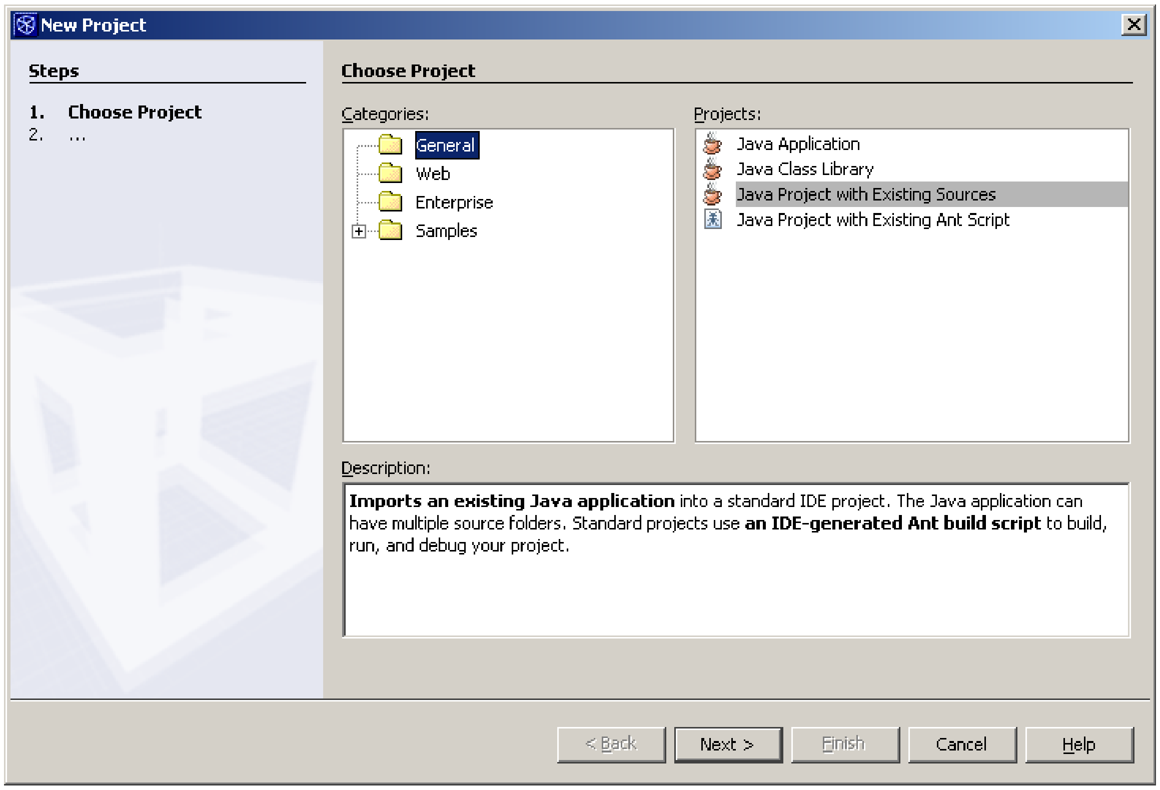
Based on the NetBeans 3.1 IDE

Important Note

We did not find NetBeans as easy to use as Eclipse or JBuilder because it does not seem to support automatic copying of non-class files (xml, txt and jpg are what we use in our example programs) to the class output directory structure. As a result, you get file I/O errors if you attempt to run our examples programs without explicitly copying txt and jpg files to the appropriate folder in the output directory. Once the java source of our example programs have been imported into a NetBeans project, you must copy some \data and \images directories that some of the example programs use. Details are given at the end of this NetBeans section. If any NetBeans experts can suggest a way of automating this procedure please let us know by sending an e-mail to support@quinn-curtis.com.

To compile and run the **QCSPCChart** for Java example programs, follow these steps.

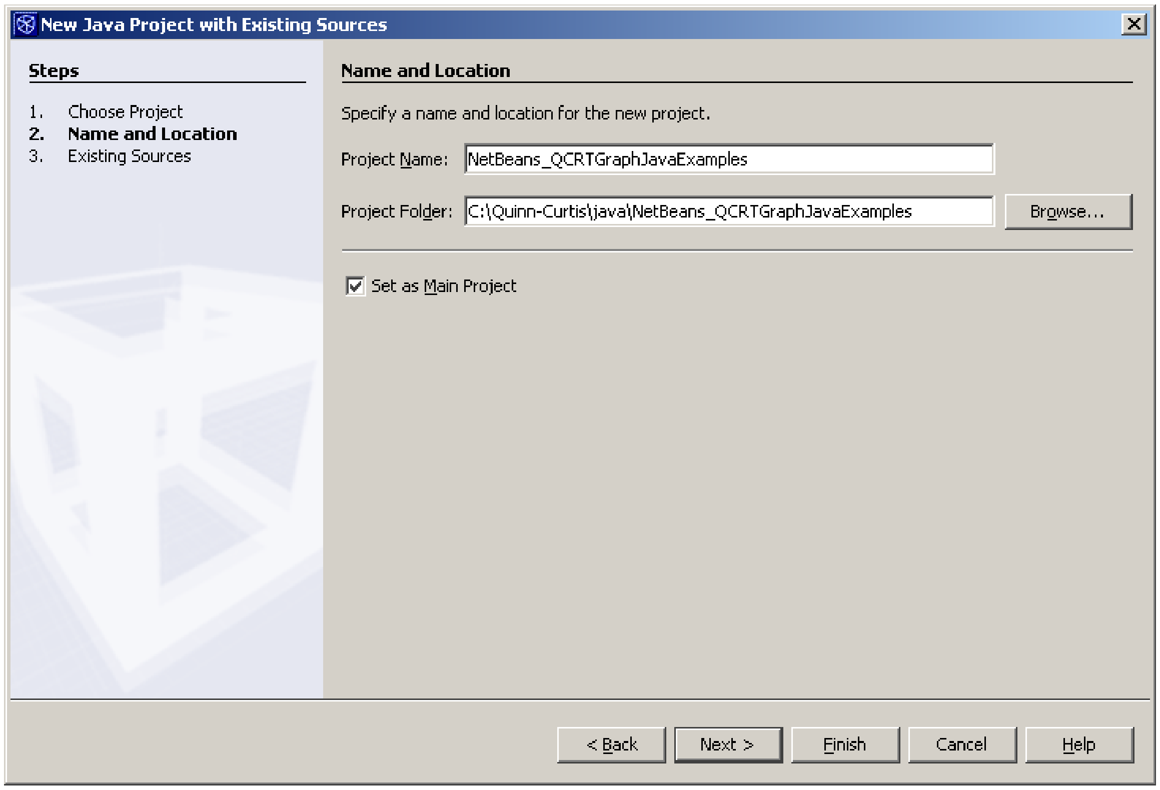
Select **File | New Project**.



Imports an existing Java application into a standard IDE project. The Java application can have multiple source folders. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Select Java Project with Existing Sources.

Click Next



New Java Project with Existing Sources

Steps

1. Choose Project
2. **Name and Location**
3. Existing Sources

Name and Location

Specify a name and location for the new project.

Project Name: NetBeans_QCRTGraphJavaExamples

Project Folder: C:\Quinn-Curtis\java\NetBeans_QCRTGraphJavaExamples

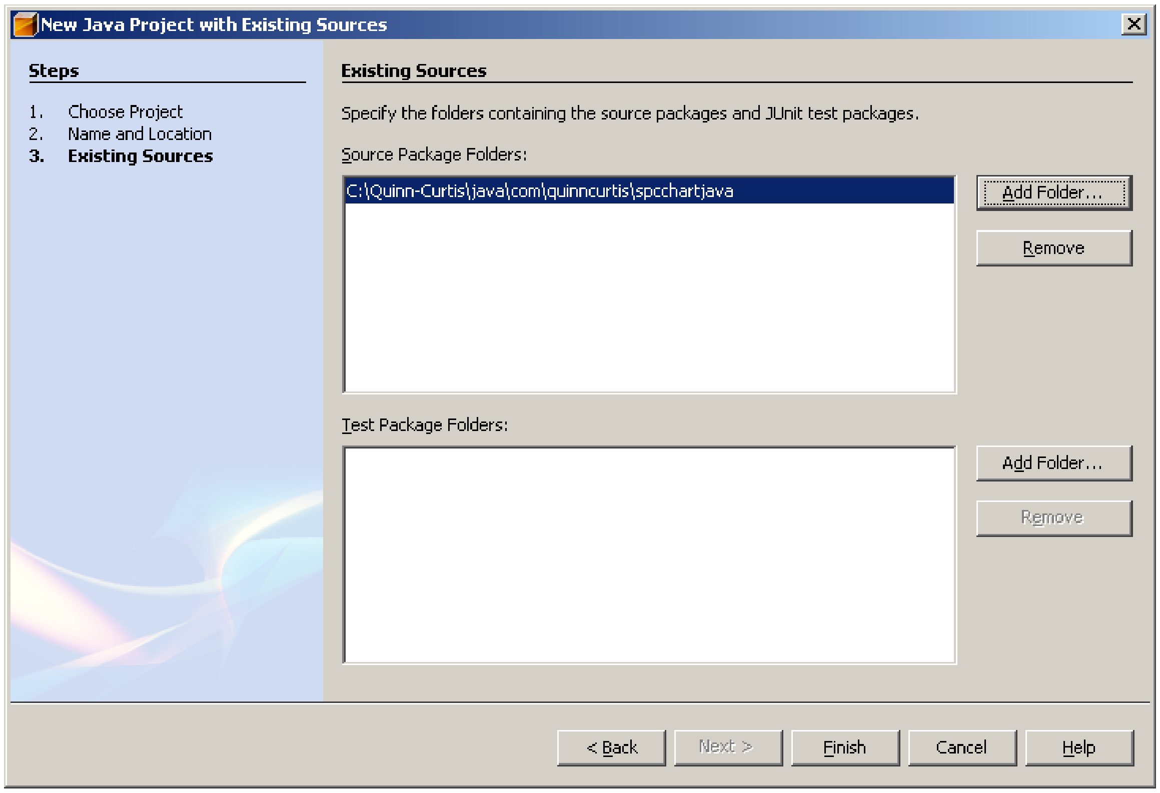
☑ Set as Main Project

We plan to import the entire java/com/quinncurtis/spcchartjava/examples directory structure, which holds all of the example programs for QCSPCChart. If you have not already done so, once you are done you may want to go back and import the QCChart2D examples from java/com/quinncurtis/chart2djava/examples as a separate project. Enter the following values:

Project Name: NetBeans_QCSPCChartJavaExamples

Project Folder: C:\Quinn-Curtis\java\NetBeans_QCSPCChartJavaExamples

Click Next.



Specify the folders containing the source packages and JUnit test packages.

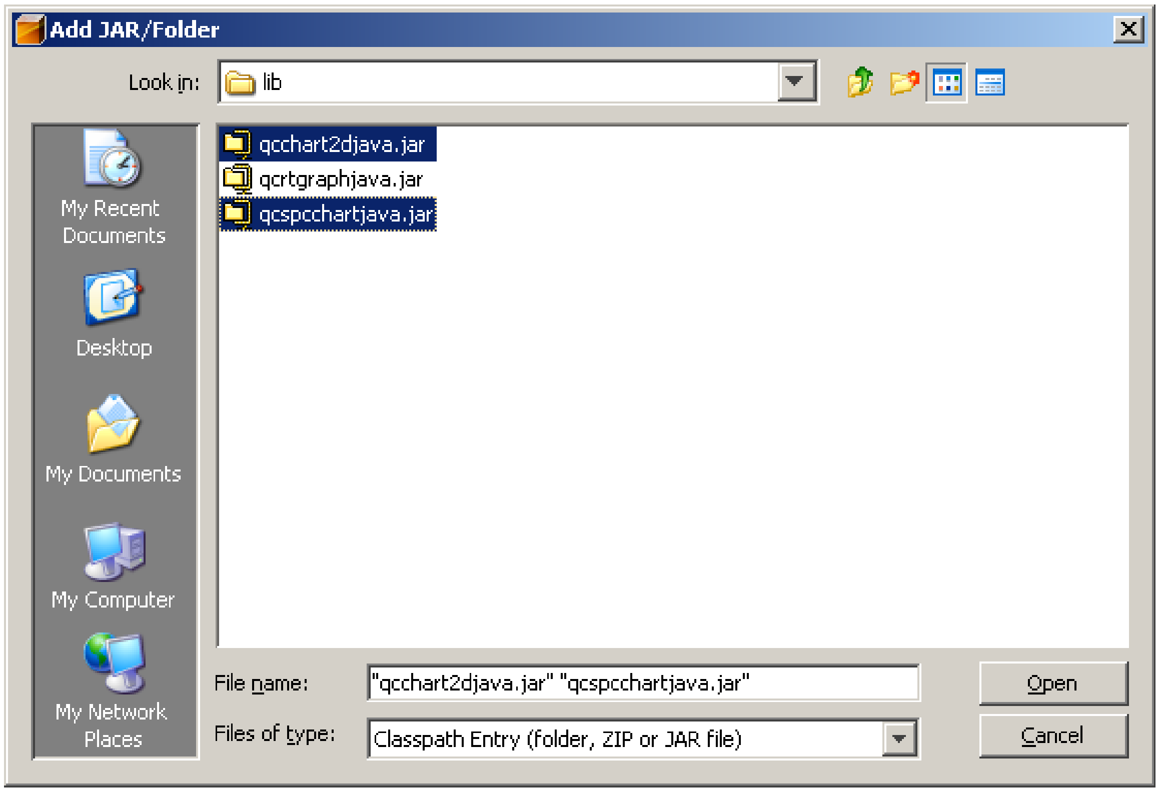
Source Package Folders: C:\Quinn-Curtis\java\com\quinncurtis\spcchartjava

Test Package Folders:

Click **Add Folder** and browse to the Quinn-Curtis/java/com/quinncurtis/spcchartjava/examples folder.

Click **Finish**.

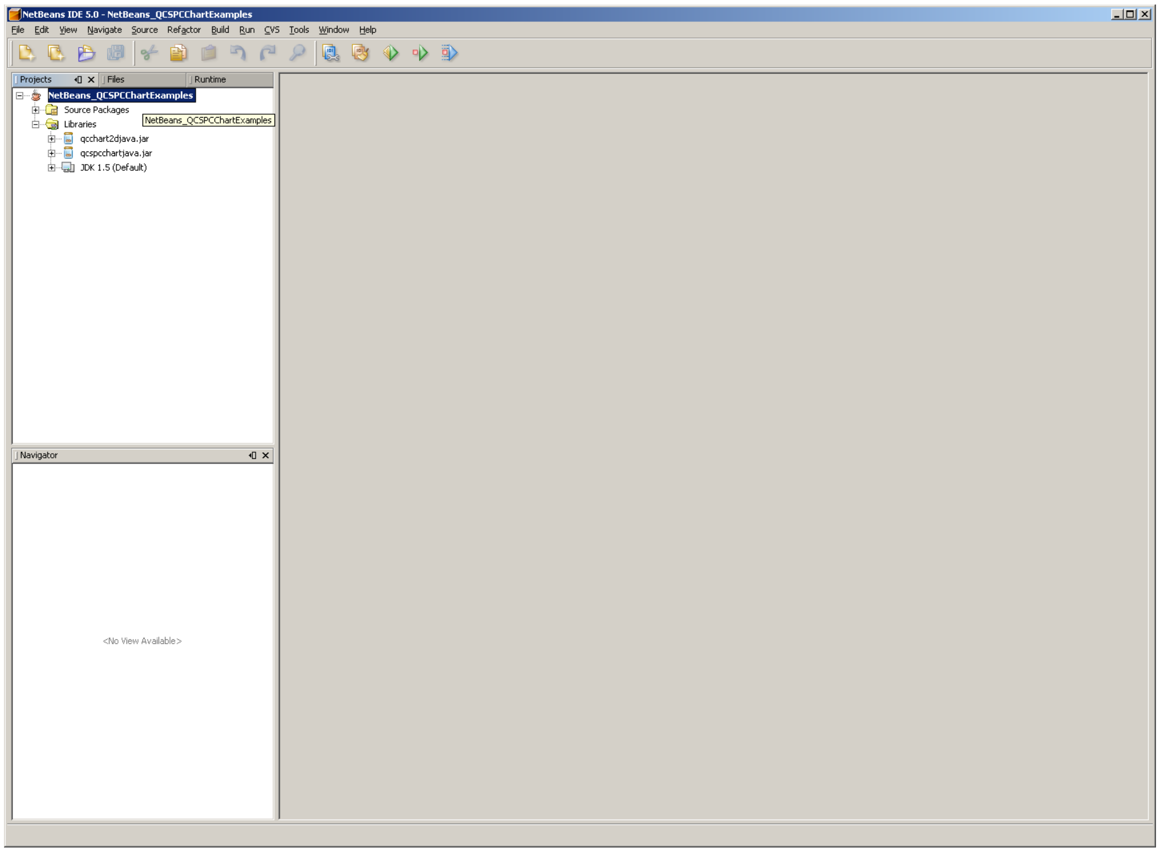
Right click on Libraries in the Project window and select **Add JAR/Folder**.



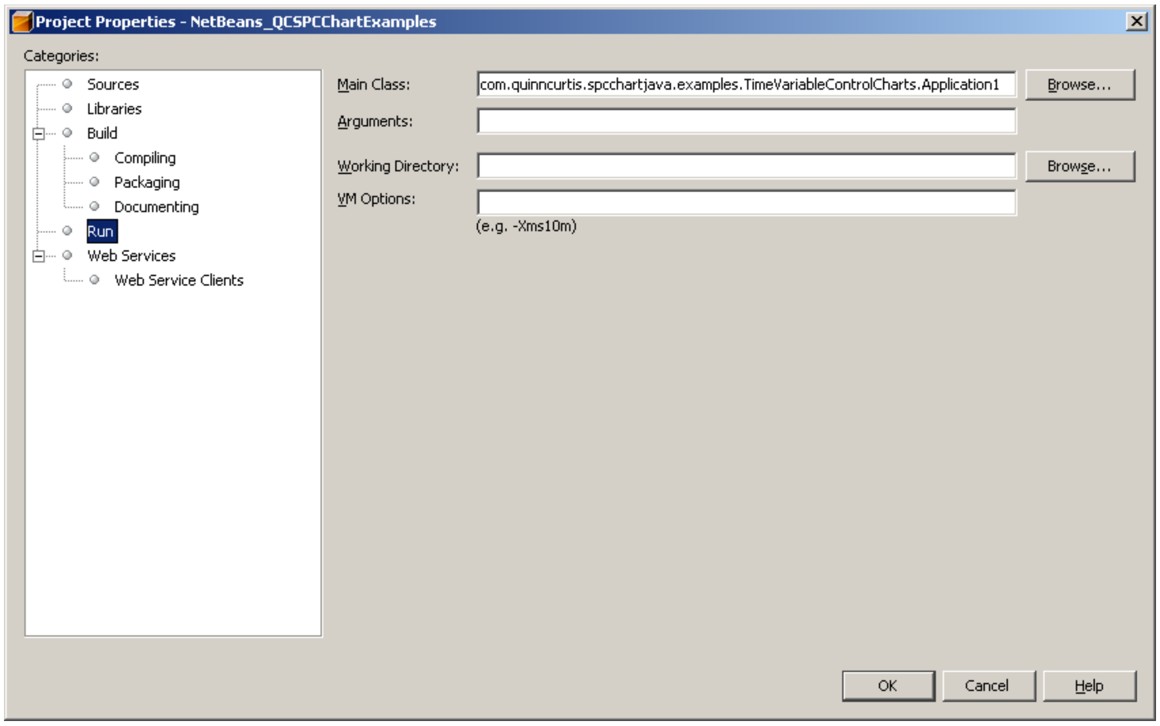
File name: "qcchart2djava.jar" "qcspcchartjava.jar"

Files of type: Classpath Entry (folder, ZIP or JAR file)

Browse to and select both the Quinn-Curtis/java/lib/**qcchart2djava.jar and** Quinn-Curtis/java/lib/**qcspcchartjava.jar** files. The **qcchart2djava.jar** and **qcspcchartjava.jar** file contains all of the QCChart2D and QCSPCChart for Java classes and documentation compressed into JAR files.


NetBeans IDE 5.0 - NetBeans_QCSPCChartExamples
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
Projects
Files
Runtime
NetBeans_QCSPCChartExamples
Source Packages
Libraries
NetBeans_QCSPCChartExamples
qcchart2djava.jar
qcspcchartjava.jar
JDK 1.5 (Default)
Navigator
<No View Available>

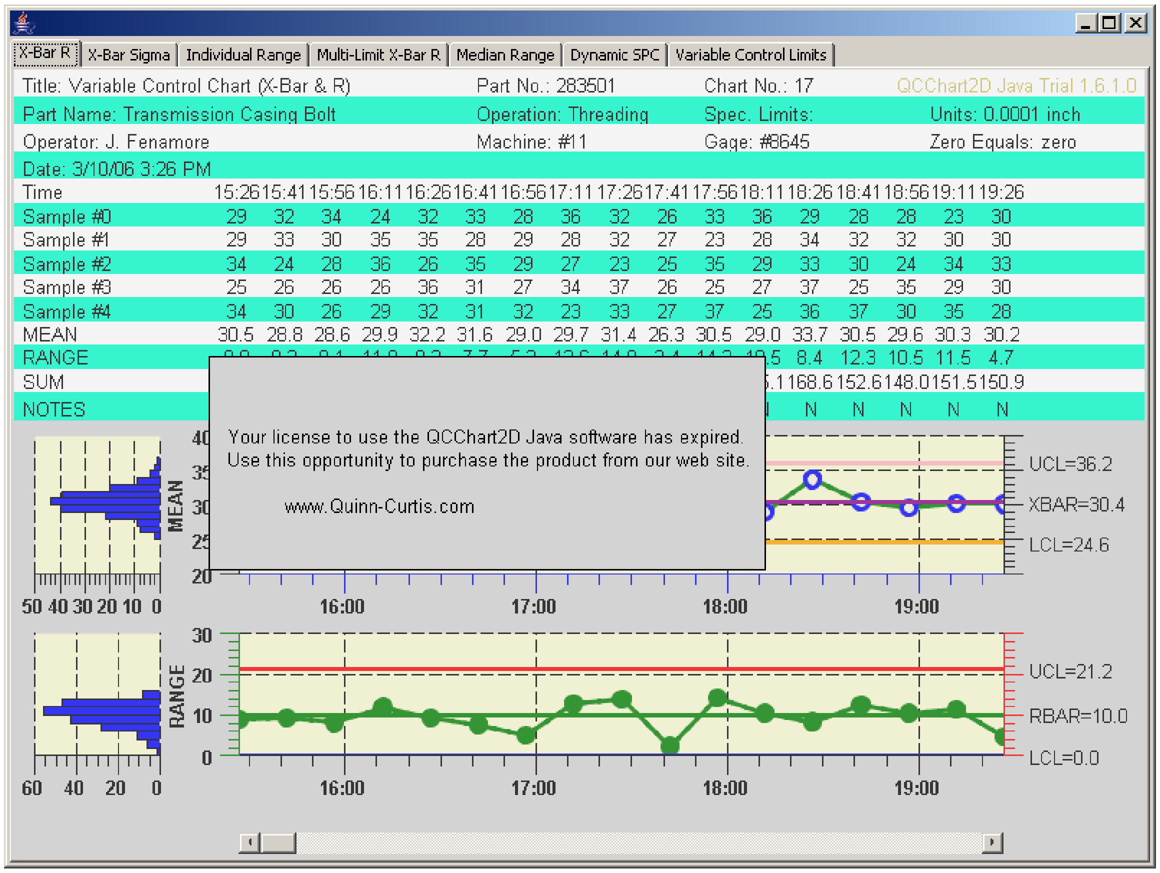
Right click on NetBeans_QCSPCChartJavaExamples and select **Properties.** On the left select **Run** under **Categories** and on the right Browse to a **Main Class** of one of the Application1 files, the Hybrid Car.Application1 file in this case. When the project runs it will execute the Main Class.



Main Class: com.quinncurtis.spcchartjava.examples.TimeVariableControlCharts.Application1
Arguments:
Working Directory:
VM Options:
(e.g. -Xms10m)

Click OK.

Right click on NetBeans_QCSPCChartJavaExamples in the Project window and select Build Project. The project should compile, probably with some warnings about “Some input files use unchecked or unsafe operations”, which is because the Java 1.5 compiler does not want you to use the collection class ArrayList (we do) without explicit type checking known as Generics (we don’t because it would make the code incompatible with earlier versions of Java).



Title: Variable Control Chart (X-Bar & R) Part No.: 283501 Chart No.: 17

Part Name: Transmission Casing Bolt Operation: Threading Spec. Limits: Units: 0.0001 inch

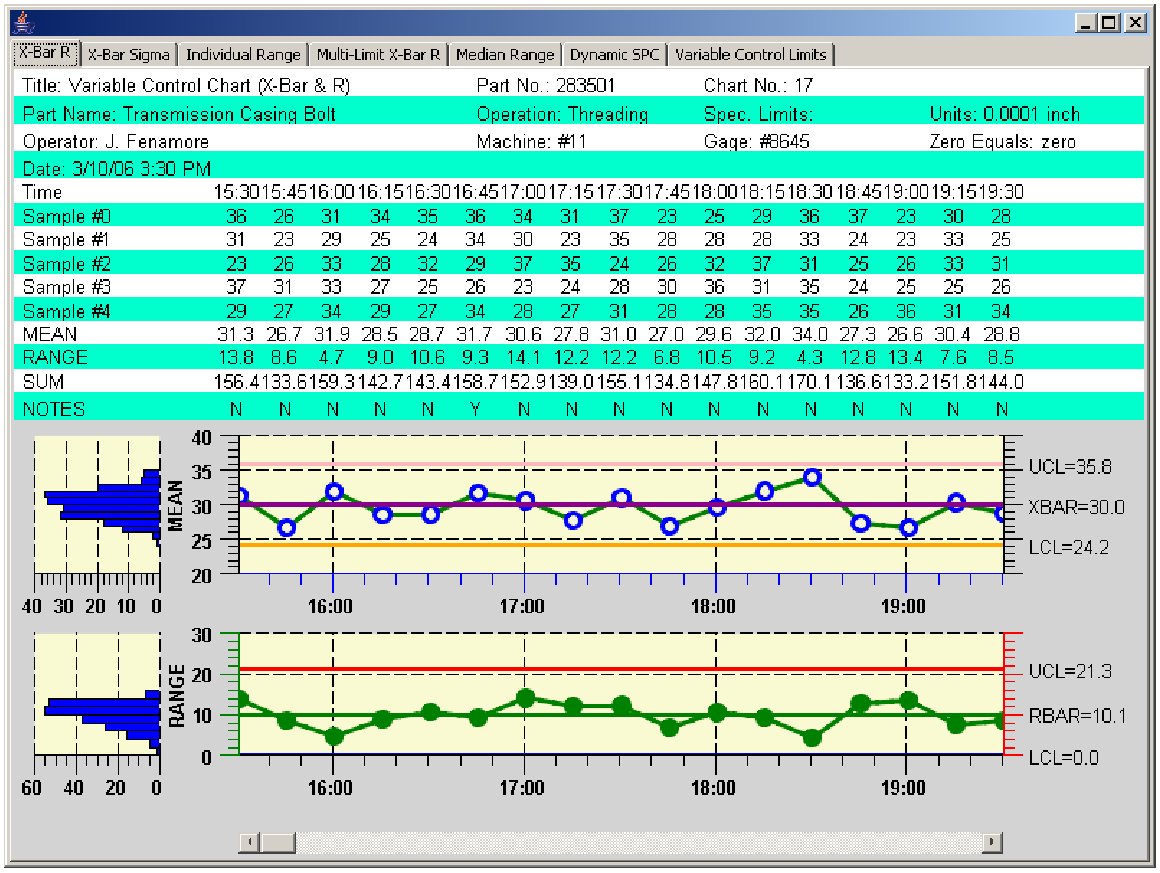
Operator: J. Fenamore Machine: #11 Gage: #8645 Zero Equals: zero

Date: 3/10/06 3:26 PM

Time	15:26	15:41	15:56	16:11	16:26	16:41	16:56	17:11	17:26	17:41	17:56	18:11	18:26	18:41	18:56	19:11	19:26
Sample #0	29	32	34	24	32	33	28	36	32	26	33	36	29	28	28	23	30
Sample #1	29	33	30	35	35	28	29	28	32	27	23	28	34	32	32	30	30
Sample #2	34	24	28	36	26	35	29	27	23	25	35	29	33	30	24	34	33
Sample #3	25	26	26	26	36	31	27	34	37	26	25	27	37	25	35	29	30
Sample #4	34	30	26	29	32	31	32	23	33	27	37	25	36	37	30	35	28
MEAN	30.5	28.8	28.6	29.9	32.2	31.6	29.0	29.7	31.4	26.3	30.5	29.0	33.7	30.5	29.6	30.3	30.2
RANGE	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	8.4	12.3	10.5	11.5	4.7
SUM	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	168.6	152.6	148.0	151.5	150.9
NOTES	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	N	N	N	N	N

Your license to use the QCChart2D Java software has expired.
Use this opportunity to purchase the product from our web site.

www.Quinn-Curtis.com



Title: Variable Control Chart (X-Bar & R) | Part No.: 283501 | Chart No.: 17

Part Name: Transmission Casing Bolt | Operation: Threading | Spec. Limits: | Units: 0.0001 inch

Operator: J. Fenamore | Machine: #11 | Gage: #8645 | Zero Equals: zero

Date: 3/10/06 3:30 PM

Time	15:30	15:45	16:00	16:15	16:30	16:45	17:00	17:15	17:30	17:45	18:00	18:15	18:30	18:45	19:00	19:15	19:30
Sample #0	36	26	31	34	35	36	34	31	37	23	25	29	36	37	23	30	28
Sample #1	31	23	29	25	24	34	30	23	35	28	28	28	33	24	23	33	25
Sample #2	23	26	33	28	32	29	37	35	24	26	32	37	31	25	26	33	31
Sample #3	37	31	33	27	25	26	23	24	28	30	36	31	35	24	25	25	26
Sample #4	29	27	34	29	27	34	28	27	31	28	28	35	35	26	36	31	34
MEAN	31.3	26.7	31.9	28.5	28.7	31.7	30.6	27.8	31.0	27.0	29.6	32.0	34.0	27.3	26.6	30.4	28.8
RANGE	13.8	8.6	4.7	9.0	10.6	9.3	14.1	12.2	12.2	6.8	10.5	9.2	4.3	12.8	13.4	7.6	8.5
SUM	156.4	133.6	159.3	142.7	143.4	158.7	152.9	139.0	155.1	134.8	147.8	160.1	170.1	136.6	133.2	151.8	144.0
NOTES	N	N	N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N

Run a different example program by selecting the projects **Properties** and changing the projects **Main Class** to one of the other examples **Application1** files (TimeAttributeControlCharts.Application1) for example.

A few of the QCChart2D example programs use image (jpg) and text (*.txt) files. These examples are CalendarData, FinancialExamples, ImageCharts and MouseListeners. The image and text files do NOT get copied over to the class output directories. So you should explicitly go into the original source directory for example program and copy the Image folder to the respective position in the class output folder (\build by default).

Copy the \data folder of the CalendarData example

Copy the \data folder of the FinancialExamples example

Copy the \images folder of the ImageCharts example

Copy the \images folder of the MouseListeners example

This is a real pain compared to Eclipse and if anyone has a better way of importing the example programs into NetBeans we would be glad to hear from you.

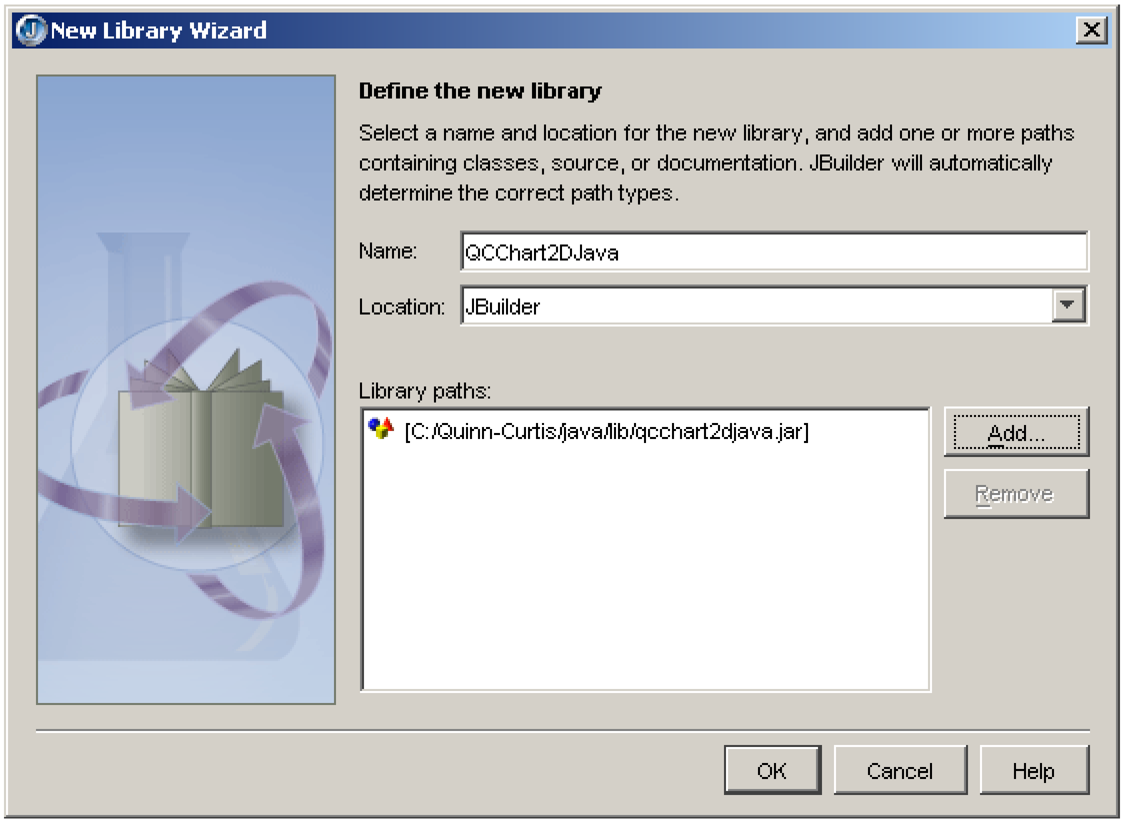
Borland JBuilder 2005 Developers Edition

These instructions assume you are using the **Borland JBuilder 2005 Developers Edition**. The JBuilder Foundation (free) editions lack the **New | Project | Project from Existing Code Wizard** that simplifies importing existing java code examples into the JBuilder project structure.

To compile and run the QCSPCChart example programs, follow these steps.

⑤ Select **Tools | Configure | Libraries**.

1. Select the **New** button from the lower left hand side, bringing up the New Library Wizard. Enter a **Name** of QCChart2DJava (use QCSPCChartJava for the qcspcchartjava.jar library) and select a location of JBuilder (the default is something else). Under Library Paths select Add and browse to and select Quinn-Curtis/java/lib/qcchart2djava.jar. Exit the wizard, taking you back to the Configure Libraries dialog.



Define the new library

Select a name and location for the new library, and add one or more paths containing classes, source, or documentation. JBuilder will automatically determine the correct path types.

Name: QCChart2DJava

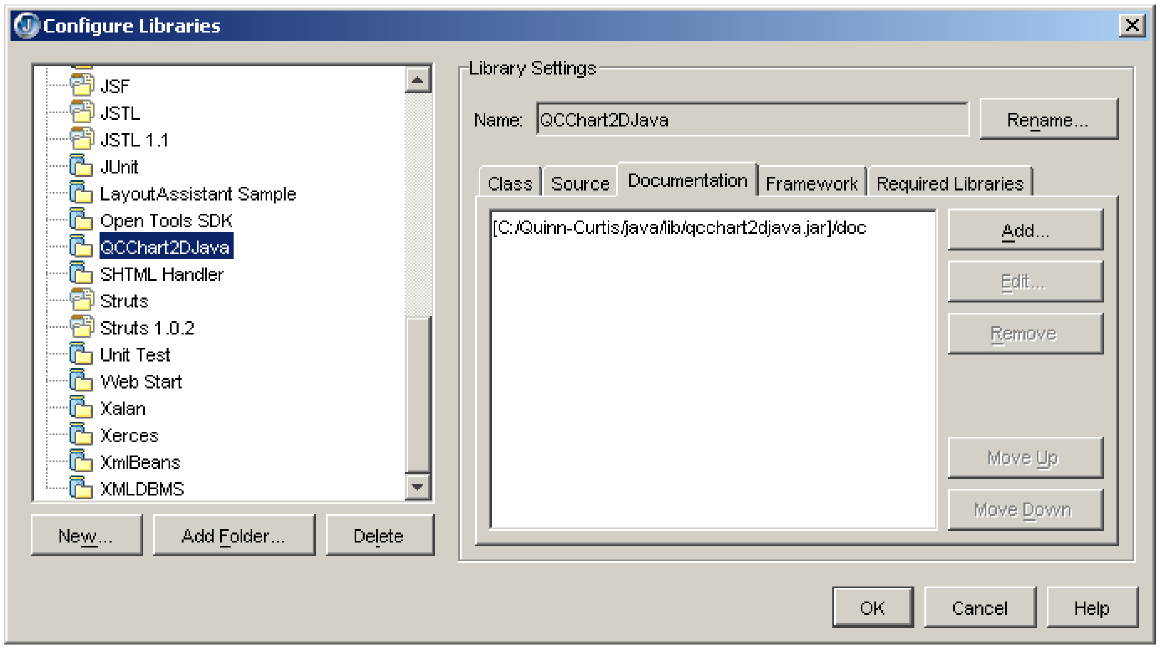
Location: JBuilder

Library paths:

[C:/Quinn-Curtis/java/lib/qcchart2djava.jar]

This adds the qcchart2djava.jar library to the standard list of JBuilder libraries that can be added to a project.

2. You should now see QCChart2DJava library (QCSPCChartJava for the qcspcchartjava.jar library) under the JBuilder node on the left. Select it there. You should see the qcchart2djava.jar file referenced on the right under the class tab.



Name: QCChart2DJava

[C:/Quinn-Curtis/java/lib/qcchart2djava.jar]/doc

3. Select the documentation tab and then Add. Browse into the qcchart2djava.jar (or qcspcchartjava.jar) file until you are able to select qcchart2djava.jar/doc (or qcspcchartjava.jar/doc). Select OK. This adds context sensitive (F1) help for the QCChart2DJava (or QCSPCChartJava) classes to the JBuilder IDE.

Repeat steps 1-3 above and add the qcspcchartjava.jar library too.

This completes the Select Tools | Configure Libraries setup. Return to the JBuilder Main Menu

⑤ Select **File | New | Project | Project for Existing Code.** For the Directory browse to and select the **Quinn-Curtis/java** directory. Enter some unique name, JBuilder_QCSPCChartJavaExamples in the example below, for the project **Name**

⑤ . Select **Next** and proceed to step 2 of the Wizard

⑤ Select the **Require Libraries** tab and select **Add**. Select the QCChart2DJava and QCSPCChartJava items under the JBuilder libraries in **the Add to Project Classpath** dialog.

This will resolve references to classes found in the **com.quinncurtis.chart2djava** and **com.quinncurtis.spcchartjava** packages.

```
import com.quinncurtis.chart2djava.*;
import com.quinncurtis.spcchartjava.*;
```

..

Select OK, returning to Step 2 of the Wizard and select Finish.

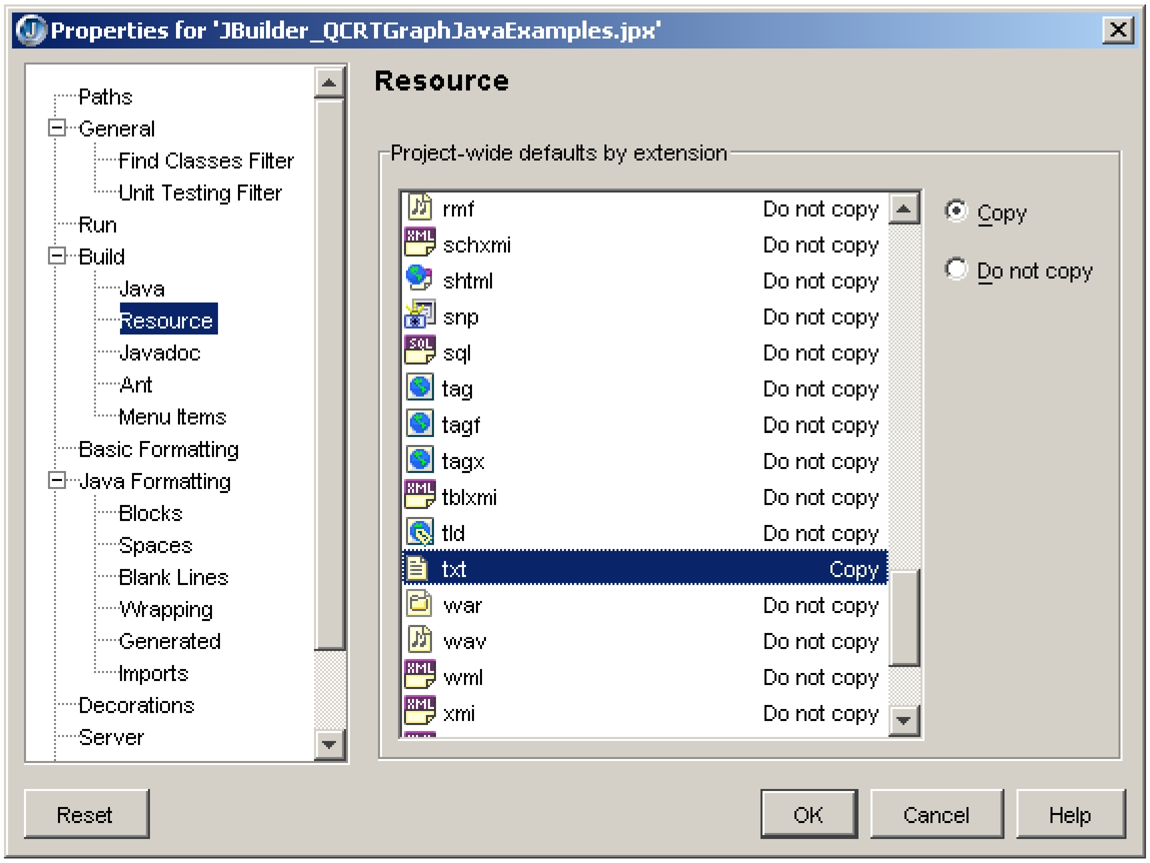
In the Project window you will see **com.quinncurtis.chart2djava.examples** and **com.quinncurtis.spcchartjava.examples** nodes. Expand those nodes and you will see all of the example programs.

Do not skip the following step

A couple of the example programs (calendardata and financialexamples) use external text data files (*.txt). JBuilder does **NOT** automatically copy these data files to the resulting class output directory, from where it executes the program. Without the data the calendardata and financialexamples will not display. In order to run these examples you must explicitly modify the project settings to support the copying of the data files to the output classes directory.

Right click on the projects *.jpx file in the project window, JBuilder_QCChart2DJava.jpx in this example) and select Properties.

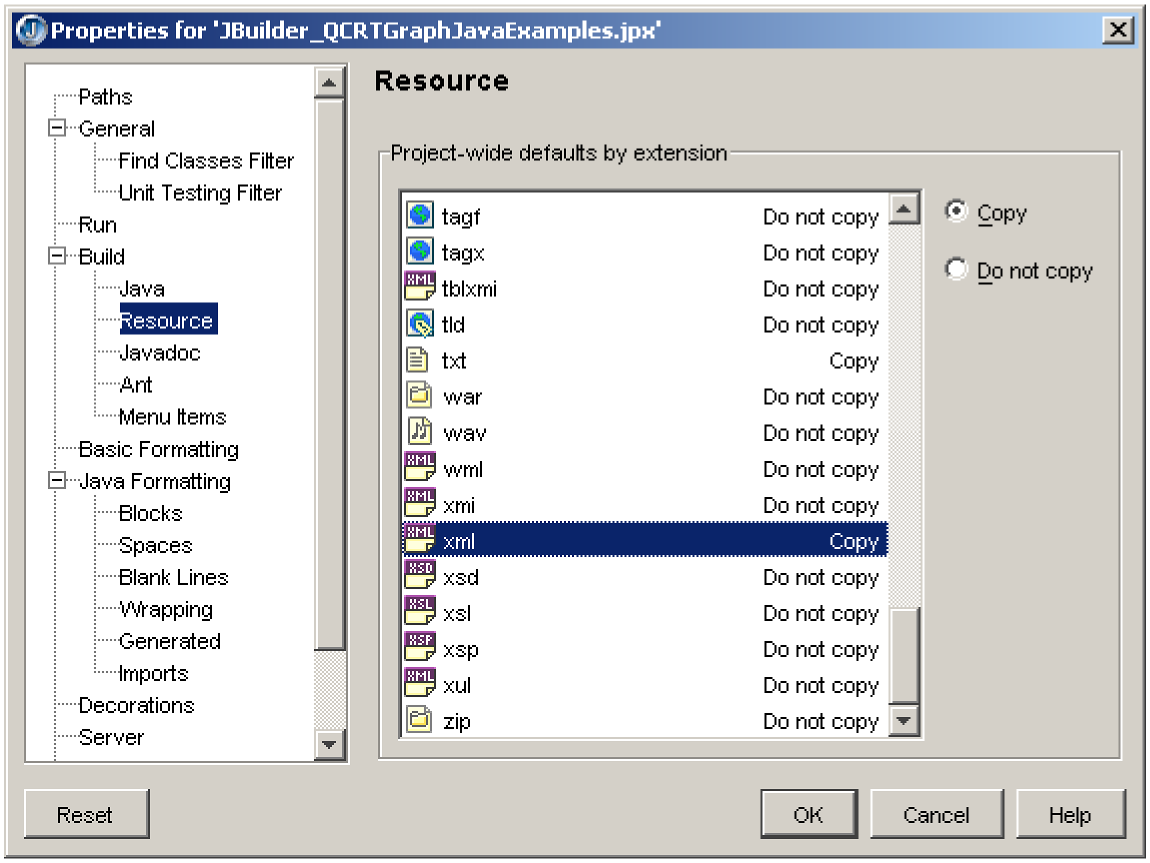
Under the **Build** node on the left select **Resources**. Scroll the list box on the right to txt (for *.txt files) and select the txt item.



rmf	Do not copy
schxmi	Do not copy
shtml	Do not copy
snp	Do not copy
sql	Do not copy
tag	Do not copy
tagf	Do not copy
tagx	Do not copy
tblxmi	Do not copy
tld	Do not copy
txt	Copy
war	Do not copy
wav	Do not copy
wml	Do not copy
xmi	Do not copy

[✓ Copy] Do not copy

Explicitly set the radio button in the upper right corner to **Copy** for that item. Select OK. From that point on all ***.txt** files in the com.quinncurtis.qcchart2djava.examples and com.quinncurtis.spcchartjava.examples directories will be copied to the output directories.



Properties for 'JBuilder_QCRTGraphJavaExamples.jpx'

- Paths
- General
  - Find Classes Filter
  - Unit Testing Filter
- Run
- Build
  - Java
  - Resource
  - Javadoc
  - Ant
  - Menu Items
- Basic Formatting
- Java Formatting
  - Blocks
  - Spaces
  - Blank Lines
  - Wrapping
  - Generated
  - Imports
- Decorations
- Server

Resource

Project-wide defaults by extension

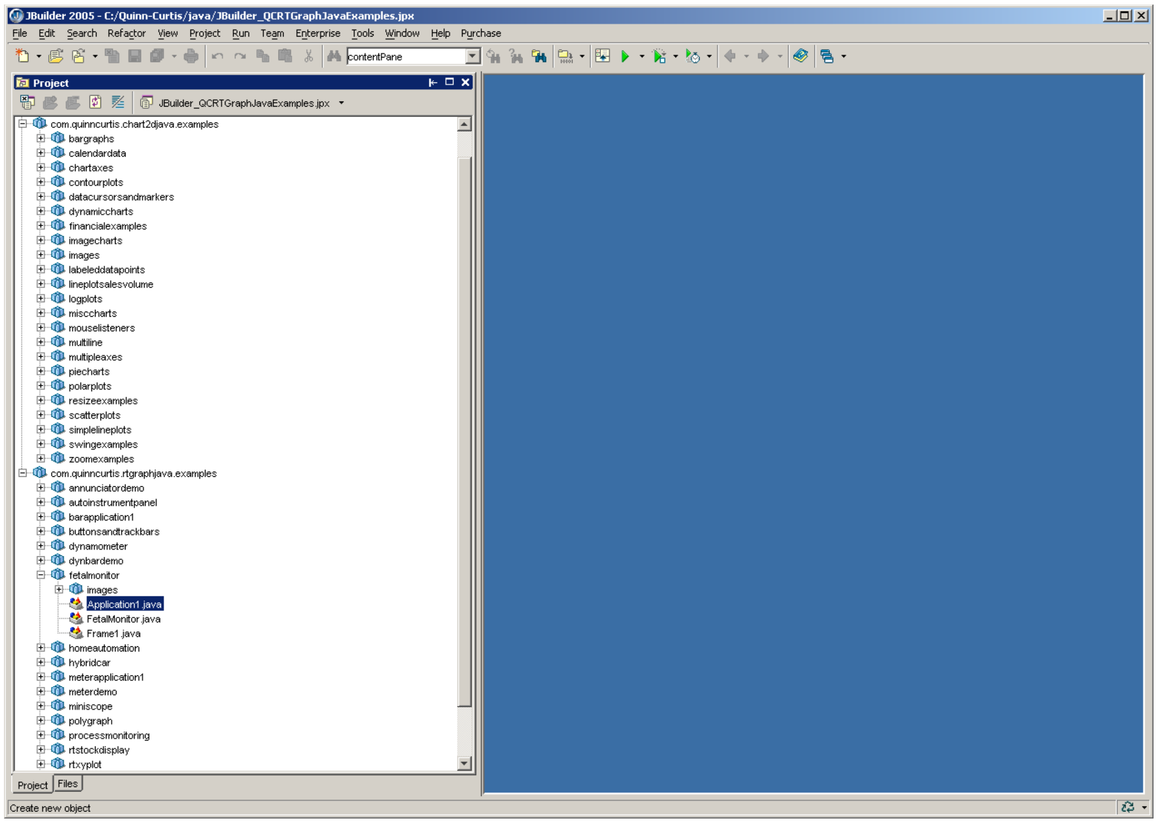
tagf	Do not copy
tagx	Do not copy
tblxmi	Do not copy
tld	Do not copy
txt	Copy
war	Do not copy
wav	Do not copy
wml	Do not copy
xmi	Do not copy
xml	Copy
xsd	Do not copy
xsl	Do not copy
xsp	Do not copy
xul	Do not copy
zip	Do not copy

[✓ Copy] Do not copy

Reset OK Cancel Help

Check that the **jpg** file extension is also marked **Copy**.

You should be able to select **Project | Make** for the project and compile all of the examples. To **Run** an individual example, expand the specific example node and select the Application1.java file under that node.



Right click on Application1.java file and select **Run using defaults**. The example program should now run.

13. Tutorial – Creating QCSPCChart for Java Applications and Applets

Java Applications

QCSPCChart for Java graphs are displayed in our **ChartView** subclass of the standard **JPanel** window class. Any program that you create that uses a **JPanel** class should be able to use our **ChartView**. You should already be a moderately experienced Java programmer before trying to use this software and should already have a collection of simple Java programs that you can experiment with. The example described below is our **SPCApplication1** example program. The main class of the program is the **Application1** class, listed below. The **Application1** class contains the static main method that creates an instance of the **Application1** class. It creates an instance of the Frame1 class that is sized for a width and height that is half the screen width and height.

```
package com.quinncurtis.spcchartjava.examples.SPCApplication1;

import java.awt.*;



 public class Applet1 extends javax.swing.JApplet {


    public void init() {

        initDemo();

    }


    public void start() {


    }


    public void initDemo()

    {


       // This skips the frame window and places the MainPanel directly

       // in the content pane of the Applet

        Container contentPane = this.getContentPane();

        SPCPanel1 spcpanel1 = new SPCPanel1();

        contentPane.add(spcpanel1, BorderLayout.CENTER);
    }
}
```

The **Frame1** class is derived from the Java **JFrame** class and provides a framework for combining menus and content panels. In this example it combines a simple menu bar with the **SPCApplication1** class, a **ChartView** derived graph. The borderLayout1 layout manager positions the menu bar at the top of the frame, and the **SPCApplication1** class in the center of the frame.

```
package com.quinncurtis.spcchartjava.examples.SPCApplication1;
import java.awt.*;
import java.awt.event.*;
import java.io.InputStream;
import javax.swing.*;
import com.quinncurtis.chart2djava.*;



/**
 * Title:  SPCApplication1 example program
 * Description:  Example program for QCRTGraph for Java  real-time graphics
software
 * Copyright:  Copyright (c) 2006
 * Company:  Quinn-Curtis, Inc.
 * @author:  Staff
 * @version  1.5
 */

public class Frame1 extends JFrame {
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();

  // Menu stuff
  JMenuBar jmenubar = new JMenuBar();
  JMenu jMenu1 = new JMenu();
  JMenuItem printMenuItem = new JMenuItem();
  JMenuItem printerSetupMenuItem = new JMenuItem();
  JMenuItem imageSetupMenuItem = new JMenuItem();

  JMenu jMenu2 = new JMenu();
  JMenuItem exitMenuItem = new JMenuItem();

  // MenuListener class found at bottom of this file
  MenuListener menulistener = new MenuListener();
```

```
SPCPanel1 spcpanel1;
ChartPrint printgraph = new ChartPrint();


//Construct the frame
public Frame1() {
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    initFrame();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

private void setupMenu() {
  // Setup menu bar
  jMenu1.setText("File");
  printMenuItem.setText("Print Graph");
  printMenuItem.addActionListener(menulistener);

  printerSetupMenuItem.setText("Printer Setup");
  printerSetupMenuItem.addActionListener(menulistener);

  imageSetupMenuItem.setText("Save As JPEG");
  imageSetupMenuItem.addActionListener(menulistener);

  jMenu2.setText("Exit");
  exitMenuItem.setText("Exit");
  exitMenuItem.addActionListener(menulistener);

  jmenubar.add(jMenu1);
  jmenubar.add(jMenu2);
  jMenu1.add(printMenuItem);
  jMenu1.add(printerSetupMenuItem);
  jMenu1.add(imageSetupMenuItem);
  jMenu2.add(exitMenuItem);
  // add menu to frame
  contentPane.add(jmenubar, BorderLayout.NORTH);
}
```

```
//Component initialization
private void initFrame() throws Exception  {
  contentPane = (JPanel) this.getContentPane();
  contentPane.setLayout(borderLayout1);

  this.setSize(new Dimension(928, 644));

  // Setup menu bar
  setupMenu();

  spcpanel1 = new SPCPanel1();
  spcpanel1.setPreferredSize(600, 400);

  contentPane.add(spcpanel1, BorderLayout.CENTER);

}


//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    System.exit(0);
  }
}

class MenuListener implements ActionListener {
  public void  actionPerformed(ActionEvent event)
  { Object object = event.getSource();

    if (object == printMenuItem)
    {
     spcpanel1.print(printgraph);
    } else
    if (object == printerSetupMenuItem)
    {
     printgraph.printDialog();
    } else
      if (object == imageSetupMenuItem)
      {
         spcpanel1.saveJPEG();
      } else
      if (object == exitMenuItem)
      {
        System.exit(0);;
      }
    }
   }

}
```

The **SPCApplication1** class is where the chart is actually defined. Since it makes calls into the **com.quinncurtis.chart2djava** and **com.quinncurtis.spcchartjava** packages, those package needs to be referenced in the import section of the class. The majority of the manual describes how to create and interact with the chart objects in the **com.quinncurtis.chart2djava** and **com.quinncurtis.spcchartjava** packages.

```
package com.quinncurtis.spcchartjava.examples.SPCApplication1;

import java.awt.*;
import java.util.*;
import com.quinncurtis.chart2djava.*;
import com.quinncurtis.spcchartjava.*;


/**
 * @author Quinn-Curtis Staff
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class SPCPanel1 extends SPCTimeVariableControlChart {
    GregorianCalendar startTime = new GregorianCalendar();
    // The time increment between adjacent subgroups
    int timeincrementminutes = 15;

      public SPCPanel1() {
        try {
            initializeChart();
        }
        catch(Exception e) {
          e.printStackTrace();
        }
      }
```

```
        public void initializeChart()
        {
            //  SPC variable control chart type
            int charttype = SPCControlChartData.MEAN_RANGE_CHART;
            // Number of samples per sub group
            int numsamplespersubgroup = 5;
            // Number of datapoints in the view
            int numdatapointsinview = 17;



            // Initialize the SPCTimeVariableControlChart
            this.initSPCTimeVariableControlChart(charttype,
                numsamplespersubgroup, numdatapointsinview, timeincrementminutes);


            this.setHeaderStringsLevel (
SPCControlChartData.HEADER_STRINGS_LEVEL1);


            // Set the strings used in the header section of the table
            this.getChartData().setTitle ( "Variable Control Chart (X-Bar & R)");
            this.getChartData().setPartNumber (  "283501");
            this.getChartData().setChartNumber( "17");
            this.getChartData().setPartName(  "Transmission Casing Bolt");
            this.getChartData().setOperation( "Threading");



            this.setGraphStartPosX ( 0.2);
        //this.GraphTopTableOffset = 0.1;

            // Display the Sampled value rows of the table
            this.setEnableInputStringsDisplay( true);
            // Display the Sampled value rows of the table
            this.setEnableCategoryValues( true);
            // Display the Calculated value rows of the table
            this.setEnableCalculatedValues( true);
            // Display the total samples per subgroup value row
            this.setEnableTotalSamplesValues( true);
            // Display the Notes row of the table
            this.setEnableNotes( true);
            // Display the time stamp row of the table
            this.setEnableTimeValues ( true);
            //this.setEnableScrollBar(true);

            SPCChartObjects primaryChart = this.getPrimaryChart();
            SPCChartObjects secondaryChart = this.getSecondaryChart();



primaryChart.getProcessVariableData().getLineMarkerPlot().setLineColor(ChartColors
.BLACK);


primaryChart.getProcessVariableData().getLineMarkerPlot().getSymbolAttributes().se
tPrimaryColor( ChartColors.BLUE);


primaryChart.getProcessVariableData().getLineMarkerPlot().getSymbolAttributes().se
tFillColor( ChartColors.LIGHTPINK);
            //  frequency histogram for both charts
            primaryChart.setDisplayFrequencyHistogram(true);
            secondaryChart.setDisplayFrequencyHistogram(true);
            this.setEnableScrollBar(true);
// used in manual
            primaryChart.getMainTitle().setTextString ("SPC Charts for .Net");
            primaryChart.getMainTitle().setTextFont ( new
Font(SPCChartBase.getDefaultChartFontString(),  Font.BOLD, 42));
            primaryChart.getMainTitle().setChartObjEnable (ChartObj.OBJECT_ENABLE);

            simulateData();


            // Calculate the SPC control limits for both graphs of the current SPC
chart (X-Bar R)
            this.autoCalculateControlLimits();
    // Scale the y-axis of the X-Bar chart to display all data and control limits
            this.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
            this.autoScaleSecondaryChartYRange();
            // Rebuild the chart using the current data and settings
            this.rebuildChartUsingCurrentData();

        }


        private void simulateData()
        {
            for (int i=0; i < 200; i++)
            {
                GregorianCalendar timestamp = (GregorianCalendar)
startTime.clone();
                // Use the ChartData sample simulator to make an array of sample
data
```

```
               DoubleArray samples =
this.getChartData().simulateMeasurementRecord(30, 10);
               // Add the new sample subgroup to the chart
               this.getChartData().addNewSampleRecord(timestamp, samples);
               // increment simulated time by timeincrementminutes minutes
               startTime.add(ChartObj.MINUTE, timeincrementminutes);
           }
       }


         public void print(ChartPrint printobj)
         {
           printobj.setPrintChartView(this);
           printobj.startPrint();
         }


        public void saveJPEG()
         { String filename = "FrequencyApplication1.jpg";
           ChartBufferedImage savegraphJPEG = new ChartBufferedImage();
           ImageFileChooser imagefilechooser = new ImageFileChooser(this);
           if (imagefilechooser.process(filename))
           {
           filename = imagefilechooser.getSelectedFilename();
           savegraphJPEG.setChartViewComponent(this);
           savegraphJPEG.render();
           savegraphJPEG.saveImageAsJPEG(filename);
           }
         }

}
```

That is the complete program. All of the other programs are setup in a similar way. Most of the other example programs place a **JTabbedPanel** in the main content panel of the frame and then place individual charts as tabs in the **JTabbedPanel**.

Standalone (without a compiler) Java Applications

If you intend to distribute standalone Java applications, you will most likely distribute it in a jar file. Setting up jar files that contain application programs that reference other jar files that contain program libraries (our qcchart2djava.jar and qcspcchartjava.jar files) can be difficult and frustrating. The best way to package Java applications is to use the Fat Jar plug-in tool available from SourceForge.net (its free),

http://sourceforge.net/projects/fjep

It simplifies packaging external JARs (our qcchart2djava.jar, qcrtgraphjava.jar and qcspcchartjava.jar files) with an application and wraps everything up into one big JAR file.

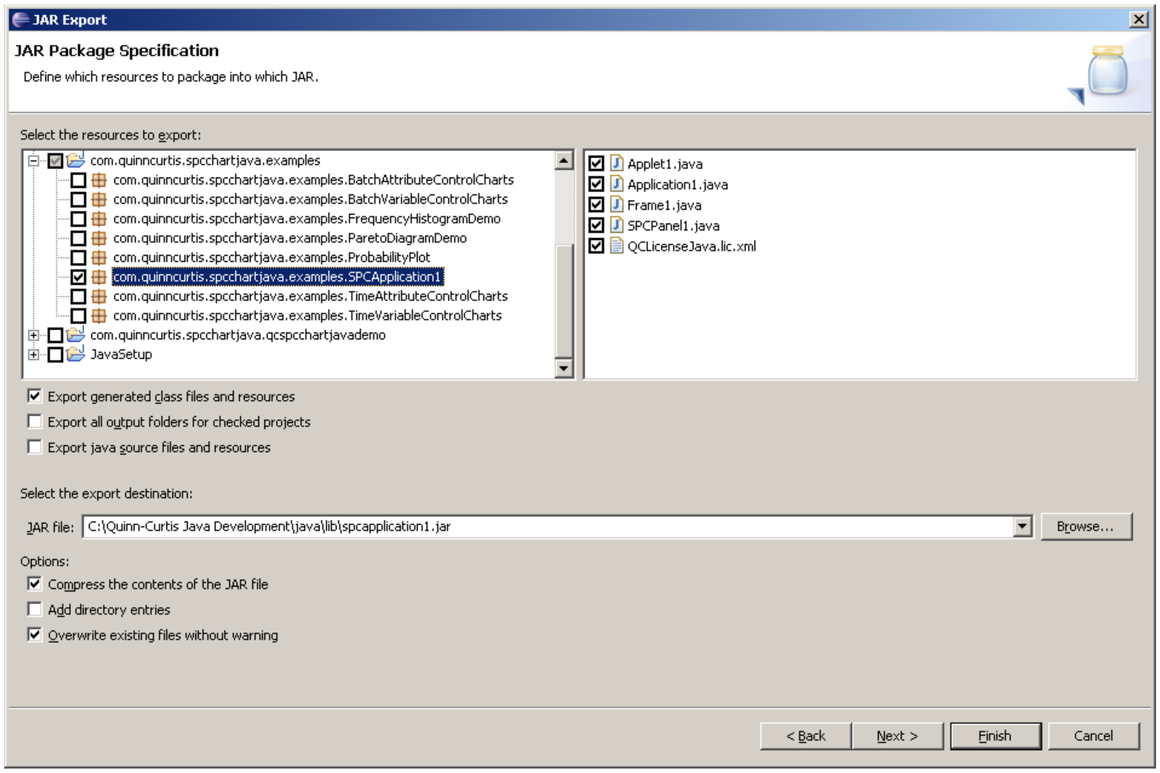
It packages a complete Eclipse project. So the best way to use it is to create your own, new, Eclipse project that includes references to our JAR libraries, and once you have that running, use the Export - Fat Jar Exporter to create the composite JAR file.

Java Applets

While Java Applications are useful if you are creating a program to run as a standalone application on a desktop, Java Applets are used to create Java programs that run in a web browser. The applet below creates the same **SPCApplication1** chart as in the previous application. The source for the Applet version is also found in the **SPCApplication1** example program directory. The **JFrame** window used in the Application example is not used here, since that would force the graph to be created in a window separate from the browser. Instead, the **SPCApplication1** graph is created in the content pane of the Applet window.

```
package com.quinncurtis.spcchartjava.examples.SPCApplication1;

import java.awt.*;

import java.io.InputStream;

import com.quinncurtis.chart2djava.*;



 public class Applet1 extends javax.swing.JApplet {


    public void init() {

        initDemo();

    }


    public void start() {


    }


    public void initDemo()

    {
        // This skips the frame window and places the MainPanel directly
        // in the content pane of the Applet
        Container contentPane = this.getContentPane();
        SPCPanel1 spcpanel1 = new SPCPanel1();
        contentPane.add(spcpanel1, BorderLayout.CENTER);
    }
}
```

The program above can be run as-is in an Applet viewer. What you really want to do is to run the program from a web browser. To do that you need to create a jar file of the applet and add the proper initialization code to an HTML page viewable from your web site. Using Eclipse you would select the File | Export option. Select the files and resources in the **SPCApplication1** directory, including any other data files and resources you are using.


JAR Export
JAR Package Specification
Define which resources to package into which JAR.

Select the resources to export:

- [✓] com.quinncurtis.spcchartjava.examples
  - [ ] com.quinncurtis.spcchartjava.examples.BatchAttributeControlCharts
  - [ ] com.quinncurtis.spcchartjava.examples.BatchVariableControlCharts
  - [ ] com.quinncurtis.spcchartjava.examples.FrequencyHistogramDemo
  - [ ] com.quinncurtis.spcchartjava.examples.ParetoDiagramDemo
  - [ ] com.quinncurtis.spcchartjava.examples.ProbabilityPlot
  - [✓] com.quinncurtis.spcchartjava.examples.SPCApplication1
  - [ ] com.quinncurtis.spcchartjava.examples.TimeAttributeControlCharts
  - [ ] com.quinncurtis.spcchartjava.examples.TimeVariableControlCharts
- [ ] com.quinncurtis.spcchartjava.qcspcchartjavademo
- [ ] JavaSetup

- [✓] Applet1.java
- [✓] Application1.java
- [✓] Frame1.java
- [✓] SPCPanel1.java
- [✓] QCLicenseJava.lic.xml

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [ ] Export java source files and resources

Select the export destination:

JAR file: C:\Quinn-Curtis Java Development\java\lib\spcapplication1.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [✓] Overwrite existing files without warning

In this case the jar file is saved using the name spcapplication1.jar. This is one of the three jars you will need in order to run the application in a web browser. The other jar files you will need are the qcchart2djava.jar and qcspcchartjava.jar files found in the \quinn-curtis\java\lib folder. For final deployment use the jar files in the \quinn-curtis\java\RedistributableFiles folders, since the RedistributableFiles version of the jar files do not include the javadoc documentation and are smaller. These jar files are uploaded to your server. The code that needs to be added to your web page will look like:

```
<html>
```

```
<body>

<p>&nbsp;</p>

<APPLET CODE =
"com.quinncurtis.spcchartjava.examples.SPCApplication1.Applet1.class"
         CODEBASE = "http://quinn-curtis.com/classes/"
         ARCHIVE = "spcapplication1.jar, qcchart2djava.jar, qcspcchartjava.jar"
WIDTH = 800 HEIGHT = 600>
  This browser does not support Java 1.2 (or greater) applets !
</APPLET> </p>


</body>

</html>
```

This HTML page (SPCApplication1.htm) can be loaded and the applet run at our web site using the following link:

http://quinn-curtis.com/SPCApplication1.Applet1.htm

Note the following:

- The CODE attribute specifies the full namespace of the main applet class, com.quinn-curtis.spcchartjava.examples.spcapplication1.Applet1.class
- The ARCHIVE attributer specifies three jars, one that contains the code of the applet (spcapplication1.jar), and the other two the QCChart2D (qcchart2djava.jar) and QCSPCChart (qcspcchartjava.jar) libraries.
- The CODEBASE attribute tells the browser in which directory to find the applet. If the applet is in the same directory as the calling HTML page the CODEBASE attribute is not necessary.
- The HEIGHT and WIDTH attribute specify the pixel dimensions of the window the applet will display in

This is the simple version of enabling an applet in a web page. The more complicated way, and the way recommended by Sun is to use the HTML <OBJECT> tag. Apparently the <APPLET> tag has been deprecated by the <OBJECT> tag in the HTML standard and Sun is encouraging web designers to use it. You can read all about it from the source at:
http://docsun.cites.uiuc.edu/sun_docs/C/solaris_9/SUNWaadm/JVPLUGINUG/p5.html

They have a program that will convert <APPLET> code to <OBJECT> code, and forces the browser to download a Java plug-in if one is not already present on the computer. It also uses the EMBED tag instead of OBJECT if the Netscape browser is used. The Netscape trick involves the use of the <COMMENT> tag. Read about that in the Sun article hyperlinked above. The converted version of the HTML applet code looks like.

```
<html>

<body>

<p>&nbsp;</p>

<OBJECT
    classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
    WIDTH = 800 HEIGHT = 600
    codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
    <PARAM NAME = CODE VALUE =
"com.quinncurtis.spcchartjava.examples.SPCApplication1.Applet1.class" >
    <PARAM NAME="codebase" VALUE="http://quinn-curtis.com/classes/" >
    <PARAM NAME = ARCHIVE VALUE = "spcapplication1.jar, qcspcchartjava.jar,
qcchart2djava.jar" >
    <PARAM NAME="type" VALUE="application/x-java-applet;version=1.4">
    <PARAM NAME="scriptable" VALUE="false">
    <COMMENT>
    <EMBED
            type="application/x-java-applet;version=1.4"
            CODE =
"com.quinncurtis.spcchartjava.examples.SPCApplication1.Applet1.class"
            CODEBASE = "http://quinn-curtis.com/classes/"
            ARCHIVE = "spcapplication1.jar, qcspcchartjava.jar, qcchart2djava.jar"
            WIDTH = 800
            HEIGHT = 600
        scriptable=false
        pluginspage="http://java.sun.com/jpi/plugin-install.html">
            <NOEMBED>
        No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
    </NOEMBED>
    </EMBED>
    </COMMENT></OBJECT>


</body>

</html>
```

This HTML page SPCApplication2.htm can be loaded and the applet run at our web site using the following link:

http://quinn-curtis.com/SPCApplication2.htm

Note that while similar, some of the attributes and tags have a different use. The best place to read about the differences is in the Sun article referenced at http://docsun.cites.uiuc.edu/sun_docs/C/solaris_9/SUNWaadm/JVPLUGINUG/p5.html

The complete source to the HTML pages are found in the SPCApplication1 examples directory, file SPCApplication1.htm and SPCApplication2.htm).

Note the following:

- The main applet class com.quinncurtis.spcchartjava.examples.SPCApplication1.Applet1.class is specified using the "code" <PARAM>
- The required jar files are specified using the "archive" <PARAM>.
- The directory that the applet is in is specified using the "codebase" <PARAM>. If the applet is in the same directory as the calling HTML page the "codebase" <PARAM> is not necessary.
- The HEIGHT and WIDTH attribute specify the pixel dimensions of the window the applet will display in
- The rest of the tags and <PARAM> values support the automatic download of the Sun Java plug-in, if the client viewing the applet needs it.
- The <EMBED> section implements an equivalent block of HTML code that is only called if a Netscape browser is used to view the applet.

Index

Adding new new data using AddNewSampleRecord . .100, 102, 103, 104, 126, 184, 185, 197, 199, 236, 237

addNewSampleRecord....98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 113, 124, 126, 143, 156, 157, 167, 168, 174, 175, 181, 182, 184, 185, 186, 196, 197, 198, 199, 212, 216, 217, 221, 222, 223, 226, 227, 234, 236, 237, 253, 334

AddNewSampleRecord...98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 113, 124, 126, 143, 156, 157, 167, 168, 174, 175, 181, 182, 184, 185, 186, 196, 197, 198, 199, 212, 216, 217, 221, 222, 223, 226, 227, 234, 236, 237, 253, 334

AIAG Rules...

AIAG Rules....vi, 241, 243, 251, 294, 300

Alarm Event Handling...................................112, 265

Alarm Event Handling. v, vi, 112, 265

Alarm highlighting..

alarm highlighting 176, 177, 179, 229, 230, 232

Alarm Highlighting.......176, 229, 263

AntennaAxes..65, 71, 73

AntennaAxes................65, 71, 72, 73

AntennaAxesLabels...72, 73

AntennaAxesLabels.................72, 73

AntennaCoordinates..61, 62

AntennaCoordinates.................61, 62

Attribute Control Chart....7, 15, 31, 35, 49, 50, 54, 94, 98, 100, 190, 191, 192, 194, 195, 212, 232, 234, 237

attribute control chart. .11, 13, 30, 46, 98, 100, 192, 193, 194, 196, 217, 223, 232, 233

Attribute Control chart. .35, 193, 194, 234

Attribute Control Chart 2, v, vi, 7, 11, 13, 15, 30, 31, 35, 41, 46, 49, 50, 54, 94, 98, 100, 190, 191, 192, 193, 194, 195, 196, 212, 217, 223, 232, 233, 234, 237

Auto-scaling classes...63, 64

AutoScale.............................1, 63, 64

AutoLogAlarmsAsNotes...

AutoLogAlarmsAsNotes.......179, 232

AutoScale...44, 92

autoScale.....106, 143, 166, 167, 212, 221, 222, 333

AutoScale 1, 3, 44, 53, 54, 63, 64, 92, 106, 128, 143, 166, 167, 212, 221, 222, 333

Axis 42, 44, 47, 48, 49, 50, 65, 66, 71, 72, 87, 92, 182, 183, 185, 235, 236

axis 1, 2, 3, 16, 38, 39, 42, 43, 52, 54, 58, 65, 66, 67, 69, 70, 71, 72, 73, 85, 86, 87, 89, 91, 106, 119, 120, 128, 143, 166, 167, 172, 182, 183, 184, 185, 191, 192, 208, 212, 220, 221, 222, 235, 236, 237, 273, 274, 275, 278, 279, 282, 293, 299, 333

Axis 1, 2, 3, vi, 16, 38, 39, 42, 43, 44, 45, 47, 48, 49, 50, 52, 53, 54, 58, 65, 66, 67, 69, 70, 71, 72, 73, 85, 86, 87, 89, 91, 92, 106, 119, 120, 128, 137, 143, 166, 167, 172, 182, 183, 184, 185, 186, 187, 191, 192, 207, 208, 212, 220, 221, 222, 235, 236, 237, 238, 239, 273, 274, 275, 278, 279, 282, 283, 293, 299, 333

AXIS.............184, 185, 235, 236, 237

Axis label classes..

AxisLabels......................1, 71, 72, 73
AxisLabels..71, 72, 92
AxisLabels............1, 3, 71, 72, 73, 92
AxisTitle..86, 87, 92
AxisTitle......86, 87, 92, 137, 207, 293
Background..5, 92
background..44, 55, 58, 65, 116, 134, 135, 204, 205
Background 5, 44, 55, 58, 65, 92, 116, 134, 135, 136, 187, 204, 205, 206, 239
BACKGROUND.134, 135, 136, 204, 205, 206
Backgrounds...65
background....................................65
Background....................................65
BarDatapointValue..92
BarDatapointValue..........................92
BufferedImage...
BufferedImage.89, 92, 284, 286, 287, 334
c-Chart..7, 15, 33, 41, 190
c-chart31, 35, 55, 100, 102, 190, 195, 196, 198, 214
c-Chart.7, 15, 31, 33, 35, 41, 55, 100, 102, 190, 195, 196, 198, 214
Calendar utilities...
ChartCalendar.................................60
Cartesian coordinates...
CartesianCoordinates..........61, 62, 63
CartesianCoordinates.........................53, 61, 62, 63, 92
CartesianCoordinates. .53, 61, 62, 63, 92
Chart Fonts.....................................136, 137, 206, 207
chart fonts............................137, 207
Chart Fonts...........136, 137, 206, 207
Chart object attributes...63
attribute..63
Attribute............................63, 64, 65
Chart Position..138, 209
Chart Position.......................138, 209
ChartAlarmEmphasisMode............................177, 230
ChartAlarmEmphasisMode...177, 230
ChartAttribute..63, 64, 65, 92
ChartAttribute......63, 64, 65, 92, 165, 258, 259, 261, 262, 263, 275
ChartCalendar........60, 90, 97, 128, 133, 201, 202, 203
ChartCalendar....60, 90, 97, 128, 133, 164, 201, 202, 203
ChartColor...
ChartColor...135, 136, 138, 165, 187, 205, 209, 238, 239, 259, 262, 263, 270, 274, 275, 278, 279, 283, 333
ChartEvent..
ChartEvent................1, 60, 63, 69, 70
ChartFont..
ChartFont.......137, 208, 289, 290, 333
ChartGradient...63
ChartImage...87, 92
ChartImage...............................87, 92
ChartLabel...86, 87, 92
ChartLabel..........................86, 87, 92
ChartObj..
ChartObj..65
ChartPlot...73, 74, 92
ChartPlot............................73, 74, 92
ChartPrint............89, 92, 284, 285, 286, 289, 329, 334
ChartPrint 89, 92, 284, 285, 286, 329, 334
printing...........................10, 284, 285
Printing..285
ChartScale..92
ChartScale....................60, 61, 62, 92
ChartShape..87, 92
ChartShape...............................87, 92
ChartSymbol...87, 88, 92
ChartSymbol.......................87, 88, 92
ChartText........................55, 86, 92, 94, 116, 138, 208
ChartText.....55, 86, 92, 94, 116, 138, 208
ChartTitle..86, 87, 92
ChartTitle...........................86, 87, 92
ChartView. .44, 59, 65, 88, 89, 92, 284, 285, 287, 327, 328
chartview.....................................287
ChartView 44, 59, 65, 88, 89, 92, 284, 285, 286, 287, 288, 327, 328, 334
Contour plotting...
ContourDataset.........................59, 60
ContourDataset...59, 60
ContourDataset.........................59, 60
Control Limit Alarms...109
control limit alarms................13, 111
Control Limit Alarms..v, 13, 109, 111
Control Limits 140, 143, 144, 145, 146, 147, 151, 155, 156, 157, 210, 212, 215, 217
control limits 2, 17, 21, 22, 31, 42, 43, 45, 94, 95, 98, 105, 106, 117, 118, 129, 137, 140, 141, 142, 143, 147, 148, 149, 150, 152, 154, 155, 156, 157, 158, 160, 161, 162, 163, 165,

166, 167, 190, 207, 210, 211, 212, 215, 217, 218, 219, 220, 221, 222, 241, 242, 245, 250, 252, 253, 254, 256, 263, 333
Control limits...97, 128, 133, 158, 218
Control Limits…2, 17, 21, 22, 31, 42, 43, 45, 94, 95, 97, 98, 105, 106, 117, 118, 128, 129, 133, 137, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 154, 155, 156, 157, 158, 160, 161, 162, 163, 165, 166, 167, 190, 207, 210, 211, 212, 215, 217, 218, 219, 220, 221, 222, 241, 242, 245, 250, 252, 253, 254, 256, 263, 333
control limits -..............................245
control limits - 2of32s..................245
control limits - 31s........................245
Cp, Cpl, Cpu, Cpk, Cpm...129
Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk.........8
Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk.......................8, 42, 116
CSV...42, 90, 92, 107
CSV............................42, 90, 92, 107
Custom Rules...
custom rules...13, 161, 177, 229, 256, 263
Custom Rules…vi, 13, 161, 177, 229, 255, 256, 257, 258, 263
Customer Support..5
customer support.............................5
Customer Support............................5
CuSum chart..................11, 16, 25, 118, 119, 153, 154
cusum chart.................................180
CuSum chart......11, 16, 25, 118, 119, 153, 154
CuSum Chart..................25, 119, 153
CUSum chart...............................117
Data logging...
data logging.................................107
Data logging................................107
Data Tooltips...
data tooltips......................1, 176, 228
Data tooltips..................................42
Data Tooltips..............1, 42, 176, 228
DataCursor..88, 89, 92
DataCursor.........................88, 89, 92
Dataset..73, 74, 92
ChartDataset.................59, 73, 74, 92
Dataset classes...59
ChartDataset..................................59
DataToolTip...
Datatooltip............................172, 173
DataToolTip. .45, 54, 88, 89, 92, 172, 173, 175, 176, 228
Developer License...
Developer License............................ii
DPMO...
dpmo...213
DPMO 2, 7, 15, 34, 41, 191, 195, 196, 213, 215
....194
Number Defects per Million 2, 7, 15, 41, 191, 194, 195
DPMO Chart..
DPMO Chart..........34, 191, 195, 213
Dunacn Rules...
Duncan Rules. vi, 241, 243, 244, 251, 294, 300
Duncan Rules...
duncan...294
Duncan.…2, 8, 12, 13, 161, 241, 243, 244, 249, 251, 256, 294, 300
DUNCAN.............250, 251, 252, 256
Eclipse...88, 89, 92
Eclipse......vi, 14, 303, 306, 307, 308, 310, 311, 320, 335, 336
ElapsedTimeAutoScale...64
ElapsedTimeAutoScale.............63, 64
ElapsedTimeAxis..65, 69, 73
ElapsedTimeAxis..........65, 69, 72, 73
ElapsedTimeAxisLabels....................................72, 73
ElapsedTimeAxisLabels...........72, 73
ElapsedTimeGroupDataset............................59, 60, 64
ElapsedTimeGroupDataset..59, 60, 64
ElapsedTimeScale...60, 61
ElapsedTimeScale....................60, 61
ElapsedTimeSimpleDataset...........................59, 60, 64
ElapsedTimeSimpleDataset 59, 60, 64
EventAutoScale...
EventAutoScale....................1, 63, 64
EventAxis...
EventAxis.................1, 65, 69, 72, 73
EventAxisLabels..
EventAxisLabels..................1, 72, 73
EventCoordinates..
EventCoordinates.................1, 61, 63
EventGroupDataset..
EventGroupDataset.........1, 59, 60, 64
EventScale...

EventScale............................1, 60, 61
EventSimpleDataset...
EventSimpleDataset........1, 59, 60, 64
EWMA chart...............................21, 22, 118, 147, 148
EWMA chart.....21, 22, 118, 147, 148
EWMA Chart.................21, 118, 147
FindObj...88, 92
FindObj....................................88, 92
FloatingBarPlot...75, 77, 92
FloatingBarPlot..................75, 77, 92
Frequency Histogram..11, 269
frequencyHistogram.....................293
FrequencyHistogram 4, 11, 13, 44, 51, 171, 187, 224, 238, 239, 267, 269, 270, 272, 273, 274, 275, 284, 286, 287, 293, 333
FrequencyHistogramChart. 4, 11, 13, 44, 51, 267, 269, 270, 272, 273, 274, 284
FrequencyHistogramChart. .4, 11, 13, 44, 51, 187, 239, 267, 269, 270, 272, 273, 274, 284
Gitlow Rules...
Getlow Rules..vi, 241, 243, 244, 251, 294, 300
gitlow...294
Gitlow......2, 8, 12, 13, 161, 241, 243, 244, 246, 248, 251, 294, 300
GITLOW..............250, 251, 252, 256
Graph object class..
GraphObj.................................63, 65
GraphObj..63, 65, 92
GraphObj...........................63, 65, 92
Grid..85, 86, 92
grid........3, 44, 51, 52, 53, 58, 86, 135
Grid..3, 44, 51, 52, 53, 58, 85, 86, 92, 135, 136
GRID..................................135, 136
Group datasets..
GroupDataset..................1, 59, 60, 64
GroupBarPlot..75, 78, 92
GroupBarPlot.....................75, 78, 92
GroupDataset...59, 60, 64, 92
GroupDataset.........1, 3, 59, 60, 64, 92
GroupPlot........................73, 75, 76, 77, 78, 79, 80, 92
GroupPlot. .73, 75, 76, 77, 78, 79, 80, 92
HistogramPlot......................................75, 78, 92, 274
HistogramPlot......75, 78, 92, 270, 274
Hughes Rules..
hughes...294
Hughes.....2, 8, 12, 13, 161, 241, 243, 244, 246, 248, 251, 254, 256, 294, 300
HUGHES.......250, 251, 252, 255, 256
Hughes Rules. vi, 241, 243, 244, 251, 254, 294, 300
Image Rendering..............................89, 284, 286, 287
BufferedImage.89, 92, 284, 286, 287, 334
Individual Range......6, 7, 15, 16, 20, 41, 48, 117, 118, 121, 146, 180
Individual Range. 6, 7, 15, 16, 20, 41, 48, 117, 118, 121, 146, 180
Initializing the SPCControlChartData Class.............95
JBuilder......................14, 303, 311, 320, 321, 322, 323
JPanel..............................43, 58, 59, 92, 327, 328, 330
Juran Rules...
juran..294
Juran....2, 5, 8, 12, 13, 161, 241, 243, 248, 251, 256, 294, 300
JURAN.................250, 251, 252, 256
Juran Rules.....vi, 241, 243, 251, 294, 300
Legend...84, 85, 92
legend......................................85, 88
Legend..........................84, 85, 88, 92
LegendItem...84, 85, 92
LegendItem.........................84, 85, 92
License File...
license file.......................................4
Linear auto-scaling..
LinearAutoScale.......................63, 64
Linear axis...
LinearAxis..........................65, 67, 72
Linear scale..
LinearScale...............................60, 61
LinearAutoScale...63, 64
LinearAutoScale.......................63, 64
LinearAxis..................................45, 65, 67, 72, 86, 92
LinearAxis.........45, 65, 67, 72, 86, 92
LinearScale...60, 61, 92
LinearScale.........................60, 61, 92
LineGapPlot..75, 79, 92
LineGapPlot.......................75, 79, 92
Log scale..
LogScale..................................60, 61
Logarithmic auto-scaling...
LogAutoScale...........................63, 64
Logarithmic axis..
LogAxis.............................65, 67, 72
LogAutoScale...63, 64
LogAutoScale...........................63, 64

LogAxis...65, 67, 72, 86, 92
LogAxis..................65, 67, 72, 86, 92
LogScale...60, 61, 92
LogScale............................60, 61, 92
MA chart....................................22, 25, 118, 119, 150
ma chart…18, 24, 117, 119, 121, 180, 181
ma Chart...............7, 15, 41, 122, 151
MA chart…21, 22, 25, 118, 119, 147, 148, 150, 151
MA Chart..........21, 22, 118, 147, 150
MAMR Chart..............................118
MR part of the MAMR (Moving Average/Moving Range Chart……151
MS part of the MAMS (Moving Average/Moving Sigma Chart……151
MAMR Chart...
MAMR Chart...........22, 23, 118, 119
MAMR Charts..
MAMR….2, 7, 15, 16, 22, 23, 41, 47, 48, 98, 117, 118, 119, 122, 151, 181, 260
MAMS..151
MAMS Chart...
MAMS Chart....................23, 24, 119
MAMS Charts..
MAMS2, 7, 15, 16, 23, 24, 41, 47, 48, 98, 117, 119, 122, 151, 181, 260
Markers...87, 88, 89, 92
marker..................................1, 82, 87
Marker. 1, 82, 83, 87, 88, 89, 92, 187, 239, 283, 333
MouseListener...45
MouseListener............45, 88, 92, 319
MoveData..88, 89, 92
MoveData..........................88, 89, 92
MoveObj..88, 92
MoveObj..................................88, 92
Moving Average/Moving Range..................................
MAMR….2, 7, 15, 16, 22, 23, 41, 47, 48, 98, 117, 118, 119, 122, 151, 181, 260
MAMS..151
Moving Average/Moving Sigma..................................
MAMS2, 7, 15, 16, 23, 24, 41, 47, 48, 98, 117, 119, 122, 151, 181, 260
MultiLinePlot...75, 79, 92
MultiLinePlot.....................75, 79, 92
Multiple SPC Control Limits..........................157, 217
Multiple SPC control limits............43
Multiple SPC Control Limits 157, 217
Nelson..
nelson...294
Nelson......2, 8, 12, 13, 161, 241, 242, 243, 246, 247, 251, 252, 256, 294, 300
NELSON..............250, 251, 252, 256
Nelson Rules...
Nelson Rules. .vi, 241, 242, 243, 251, 252, 294, 300
Nelson Rules. .2, 8, 12, 13, 161, 241, 242, 243, 246, 247, 250, 251, 252, 256, 294
NetBeans...................14, 303, 311, 313, 316, 317, 320
Notes Tooltips..172, 225
notes tooltips................175, 176, 228
Notes Tooltips.....172, 175, 176, 225, 228
NotesLabel...45, 54, 138, 209
NotesLabel 45, 54, 136, 138, 206, 209
NotesToolTip.....................................45, 54, 176, 228
NotesToolTip….45, 54, 175, 176, 228
np-Chart...32, 190
np-chart....31, 55, 100, 101, 190, 195, 196, 197, 214
np-Chart................................32, 190
Number Defects per Million..
Number Defects per Million. 2, 7, 15, 299
Number Defects Per Million.........215
Number Defects per Million. .2, 7, 15
Numeric axis labels...
NumericAxisLabels..................71, 72
NumericAxisLabels......................................71, 72, 92
NumericAxisLabels............71, 72, 92
NumericLabel..........55, 86, 87, 92, 116, 138, 208, 209
NumericLabel......55, 86, 87, 92, 116, 138, 208, 209
OHLCPlot...75, 80, 92
OHLCPlot..........................75, 80, 92
p-Chart.................................4, 31, 32, 41, 49, 50, 190
p-chart......31, 55, 100, 101, 102, 190, 195, 196, 197, 213, 214
p-Chart.4, 31, 32, 41, 49, 50, 55, 100, 101, 102, 190, 195, 196, 197, 213, 214
Pareto Chart..
paretoChart.................................293
ParetoChart..12, 13, 44, 52, 267, 280, 281, 282, 284, 293

ParetoChart......12, 13, 44, 52, 267, 280, 281, 282, 284
paretoChart..................................293
ParetoChart..12, 13, 44, 52, 267, 280, 281, 282, 284, 293
Physical coordinates
PhysicalCoordinates......61, 62, 64, 65
PhysicalCoordinates.............45, 54, 61, 62, 64, 65, 92
PhysicalCoordinates....45, 54, 61, 62, 64, 65, 92
Polar axes
PolarAxes.....................65, 70, 71, 73
Polar axis labels
PolarAxesLabels.......................71, 73
Polar coordinates
PolarCoordinates.......................61, 62
PolarAxes..65, 70, 73
PolarAxes.....................65, 70, 71, 73
PolarAxesLabels...71, 73
PolarAxesLabels.......................71, 73
PolarCoordinates...61, 62
PolarCoordinates.......................61, 62
Pp, Ppu, Ppl, and Ppk..129
Probability Chart
probabilityChart...................293, 294
ProbabilityChart. .12, 13, 44, 53, 267, 276, 277, 278, 284, 293, 294
ProbabilityChart.....12, 13, 44, 53, 267, 276, 277, 278, 284
probabilityChart...................293, 294
ProbabilityChart. .12, 13, 44, 53, 267, 276, 277, 278, 284, 293, 294
ProbabilityCoordinates..................................45, 53, 54
ProbabilityCoordinates........45, 53, 54
ProbabilityScale..45, 53
ProbabilityScale.......................45, 53
ProbabilitySigmaAxis...................................45, 53, 54
ProbabilitySigmaAxis.........45, 53, 54
process capability..46
Process Capability...8, 42
process capability 8, 42, 46, 116, 129, 130, 172, 173, 262
Process Capability...8, 129, 130, 131, 132, 263
Process Capability Ratios...129
Process Performance
Process Performance.....129, 131, 132
Process Performance Indices...................................129
QCSPCChartNet.DLL
QCSPCChartJava.........................289
Redistributable License...4
Redistributable License................ii, 4
Regionalization
regionalization..............................289
Regionalization.........................2, 289
Rule Templates
rule templates...........................2, 161
Rule Templates.............................245
Scale classes..60, 61, 62
ChartScale..........................60, 61, 62
Scatter Plots..27, 169, 223
scatter plots........27, 44, 58, 73, 81, 82
Scatter plots...................................43
Scatter Plots27, 43, 44, 58, 73, 81, 82, 169, 223
Scrollbar
scrollbar....................26, 35, 170, 223
ScrollBar......170, 184, 185, 223, 236, 238, 332, 333
Simple datasets
SimpleDataset.................1, 59, 60, 64
SimpleBarPlot...82, 83, 92
SimpleBarPlot....................82, 83, 92
SimpleDataset..59, 60, 64, 92
SimpleDataset........1, 3, 59, 60, 64, 92
SimpleLineMarkerPlot..................................82, 83, 92
SimpleLineMarkerPlot........82, 83, 92
SimpleLinePlot...82, 84, 92
SimpleLinePlot...................82, 84, 92
SimplePlot............................73, 82, 83, 84, 88, 89, 92
SimplePlot...73, 82, 83, 84, 88, 89, 92
SimpleScatterPlot..82, 84, 92
SimpleScatterPlot...............82, 84, 92
SPCArrayStatistics..54
SPCArrayStatistics.........................54
SPCBatchAttributeControlChart..2, 11, 13, 44, 46, 49, 190, 192, 232, 233, 284
SPCBatchAttributeControlChart2, 11, 13, 44, 46, 49, 101, 102, 190, 192, 196, 198, 207, 232, 233, 238, 284
SPCBatchVariableControlChart. .2, 11, 13, 44, 46, 47, 95, 117, 118, 120, 179, 180, 181, 207, 284
SPCBatchVariableControlChart2, 11, 13, 44, 46, 47, 95, 117, 118, 120, 137, 148, 152, 179, 180, 181, 187, 207, 284
SPCCalculatedValueRecord. .45, 46, 94, 95, 110, 112, 115, 116
SPCCalculatedValueRecordv, 45, 46, 94, 95, 110, 112, 115, 116, 164, 169, 259, 260, 261, 262
SPCChartBase......44, 46, 95, 109, 115, 123, 177, 178, 179, 195, 230, 231, 232, 284, 333

SPCChartBase…44, 46, 95, 109, 115, 123, 177, 178, 179, 194, 195, 230, 231, 232, 239, 284, 333
SPCChartObjects.....45, 128, 137, 138, 155, 156, 159, 184, 185, 187, 207, 208, 215, 216, 217, 219, 220, 235, 236, 237, 238, 239, 332, 333
SPCChartObjects...45, 128, 137, 138, 155, 156, 159, 165, 172, 184, 185, 187, 207, 208, 215, 216, 217, 219, 220, 235, 236, 237, 238, 239, 250, 251, 253, 255, 257, 258, 259, 261, 262, 263, 332, 333
SPCControlChartData......45, 94, 95, 96, 97, 101, 102, 107, 108, 109, 110, 111, 112, 113, 114, 115, 121, 122, 127, 128, 132, 133, 141, 142, 149, 152, 154, 180, 181, 192, 196, 197, 201, 202, 203, 204, 211, 221, 222, 223, 227, 232, 234, 265, 332
sPCControlChartData...................291
SPCControlChartData…v, 45, 94, 95, 96, 97, 101, 102, 107, 108, 109, 110, 111, 112, 113, 114, 115, 121, 122, 127, 128, 132, 133, 141, 142, 149, 152, 154, 164, 168, 169, 180, 181, 192, 196, 197, 201, 202, 203, 204, 211, 221, 222, 223, 227, 232, 234, 253, 265, 291, 332
SPCControlLimitAlarmArgs 45, 56, 94, 112, 113, 114, 115, 265
SPCControlLimitAlarmArgs...45, 56, 94, 112, 113, 114, 115, 164, 168, 265
SPCControlLimitRecord45, 46, 94, 95, 109, 110, 111, 112, 113, 114, 141, 142, 159, 160, 211, 219, 220
SPCControlLimitRecord…45, 46, 94, 95, 109, 110, 111, 112, 113, 114, 141, 142, 158, 159, 160, 164, 169, 211, 219, 220, 250, 251, 252, 254, 255, 256
SPCControlParameters.......................................45, 54
SPCControlParameters.............45, 54
SPCControlPlotObjectData...
SPCControlPlotObjectData.258, 259, 261, 262, 263
SPCGeneralizedTableDisplay......45, 55, 94, 116, 134, 135, 136, 138, 205, 206, 207, 208
SPCGeneralizedTableDisplay…v, 45, 55, 94, 116, 134, 135, 136, 138, 205, 206, 207, 208
SPCTimeAttributeControlChart……11, 13, 44, 46, 50, 190, 192, 193, 200, 233, 238, 284
SPCTimeAttributeControlChart....11, 13, 44, 46, 50, 101, 102, 103, 190, 192, 193, 196, 197, 198, 199, 200, 201, 207, 208, 233, 238, 284
SPCTimeVariableControlChart. 11, 12, 13, 44, 46, 48, 95, 97, 105, 111, 113, 117, 118, 120, 121, 122, 127, 137, 149, 152, 154, 180, 208, 284, 331, 332
SPCTimeVariableControlChart….11, 12, 13, 44, 46, 48, 95, 97, 105, 109, 111, 113, 114, 117, 118, 120, 121, 122, 127, 137, 149, 152, 154, 163, 180, 187, 208, 284, 331, 332
Specification Limits..
specification limits 129, 158, 165, 219
Specification limits.......................165
Specification Limits...........2, 42, 165
StackedBarPlot...75, 78, 92
StackedBarPlot...............3, 75, 78, 92
StackedLinePlot..75, 80, 92
StackedLinePlot.................75, 80, 92
StandardLegend...85, 92
StandardLegend........................85, 92
String axis labels..
StringAxisLabels......................71, 72
StringAxisLabels..71, 72, 92
StringAxisLabels................71, 72, 92
StringLabel................45, 55, 86, 87, 92, 116, 138, 209
StringLabel….45, 55, 86, 87, 92, 116, 136, 138, 184, 185, 206, 209, 235, 236, 237
Table Background Colors..134
Table Background Colors.............134
Table Fonts...136, 206
table fonts............................136, 206
Table Fonts...........................136, 206
Table Strings...132
table strings...................................96
Table Strings..........................96, 132
TableAlarmEmphasisMode.............178, 179, 231, 232
TableAlarmEmphasisMode.178, 179, 231, 232
Templates...7, 138, 208, 331
templates....2, 7, 8, 13, 15, 26, 35, 42, 46, 138, 161, 208, 209, 246, 255
Templates…2, vi, 7, 8, 13, 15, 26, 35, 42, 46, 138, 161, 208, 209, 245, 246, 255, 331
TickMark...90, 91, 92
TickMark...........................90, 91, 92
Time auto-scaling...
TimeAutoScale.........................63, 64
Time axis..
TimeAxis......................65, 69, 72, 73

Time axis labels
TimeAxisLabels........................72, 73
Time coordinates
TimeCoordinates.......................61, 62
Time scale
TimeScale.................................60, 61
Time/Date group datsets
TimeGroupDataset.............59, 60, 64
Time/Date simple datasets
TimeSimpleDataset............59, 60, 64
TimeAutoScale......63, 64
TimeAutoScale.........................63, 64
TimeAxis......65, 69, 73, 86, 92
TimeAxis.......3, 65, 69, 72, 73, 86, 92
TimeAxisLabels......72, 73, 92
TimeAxisLabels.............3, 72, 73, 92
TimeCoordinates......61, 62, 92
TimeCoordinates............3, 61, 62, 92
TimeGroupDataset......59, 60, 64, 92
TimeGroupDataset....3, 59, 60, 64, 92
TimeLabel......55, 86, 87, 92, 116, 138, 208, 209
TimeLabel 3, 55, 86, 87, 92, 116, 136, 138, 206, 208, 209
TimeScale......60, 61, 92
TimeScale.......................3, 60, 61, 92
TimeSimpleDataset......59, 60, 64, 92
TimeSimpleDataset...3, 59, 60, 64, 92
ToolTips
Datatooltip............................172, 173
DataToolTip. .45, 54, 88, 89, 92, 175, 176, 228
Trial License
Trial License....................................ii
u-Chart......4, 7, 15, 33, 34, 41, 191
u-chart......12, 13, 31, 34, 35, 55, 102, 103, 190, 193, 195, 198, 214, 233
u-Chart4, 7, 12, 13, 15, 31, 33, 34, 35, 41, 55, 102, 103, 190, 191, 193, 195, 198, 214, 233
User coordinates
UserCoordinates.......................61, 62
UserCoordinates......61, 62, 92
UserCoordinates.................61, 62, 92
Variable Control Chart. 7, 8, 15, 16, 17, 26, 35, 47, 48, 97, 98, 117, 120, 123, 127, 133, 143, 179, 186, 192, 332
variable control chart...11, 13, 18, 46, 96, 98, 99, 111, 117, 120, 121, 122, 123, 124, 125, 148, 149, 152, 154, 157, 162, 173, 180, 217, 332
Variable control chart.............16, 119
Variable Control chart....................15
Variable Control Chart...2, v, vi, 7, 8, 11, 13, 15, 16, 17, 18, 26, 35, 41, 46, 47, 48, 96, 97, 98, 99, 111, 117, 119, 120, 121, 122, 123, 124, 125, 127, 133, 143, 148, 149, 152, 154, 157, 162, 173, 179, 180, 186, 192, 217, 291, 298, 332
variable sample subgroup. .12, 102, 103, 117, 197, 198
variable sample subgroup......12, 102, 103, 117, 197, 198
Variable Sample Subgroup..102, 103, 197, 198
Visual Basic.....14, 303, 306, 307, 308, 310, 311, 320, 336
WE rules......12
WE rules................................12, 162
WE Rules......................161, 162, 242
WECO
weco...294
WECO. 8, 12, 13, 241, 242, 243, 246, 247, 250, 251, 252, 254, 256, 294, 300
WECO+Supplemental..................247
....246
Western Electric (WE) Runtime Rules......160
Western Electric rules
Western Electric Rules..161, 241, 242
Westgard
westgard......................................242
Westgard.......................242, 245, 249
Westgard Rules
Westgard Rules...............vi, 242, 245
Working coordinates
WorkingCoordinates................61, 62
WorkingCoordinates......61, 62, 92
WorkingCoordinates..........61, 62, 92
World coordinates
WorldCoordinates....................61, 62
WorldCoordinates......61, 62, 92
WorldCoordinates..............61, 62, 92
X-Bar R. .6, 7, 8, 11, 12, 13, 15, 16, 17, 19, 21, 22, 41, 47, 55, 98, 117, 118, 121, 123, 143, 144, 180, 184, 185, 333
X-Bar R 6, 7, 8, 11, 12, 13, 15, 16, 17, 19, 21, 22, 41, 47, 55, 98, 117, 118, 121, 123, 143, 144, 162, 164, 180, 184, 185, 333
X-Bar Sigma 4, 7, 8, 11, 12, 13, 15, 16, 17, 18, 19, 41, 55, 98, 99, 117, 121, 123, 124, 125, 143, 145, 180
X-Bar Sigma....4, 7, 8, 11, 12, 13, 15, 16, 17, 18, 19, 41, 55, 98, 99, 117,

121, 122, 123, 124, 125, 143, 145, 180, 181

X-R...7, 8, 11, 12, 13, 15, 16, 20, 41, 55, 117, 118, 146
X-R....7, 8, 11, 12, 13, 15, 16, 20, 41, 55, 117, 118, 146
XAxisLabelRotation...183
XAxisLabelRotation.....................183
XAxisStringLabelMode...........184, 185, 235, 236, 237
XAxisStringLabelMode......184, 185, 235, 236, 237
Zooming...3

zoom.....................................3, 88, 89

Zoom...............................3, 88, 89, 92